--1a

--in

-- Get employees who work in departments 10, 20, or 30

SELECT first\_name, department\_id

FROM employees

WHERE department\_id IN (10, 20, 30);

-- Get employees working in same departments as job\_id = 'IT\_PROG'

SELECT first\_name, department\_id

FROM employees

WHERE department\_id IN (

SELECT department\_id

FROM employees

WHERE job\_id = 'IT\_PROG'

);

--any

-- Get employees whose salary is greater than \*\*any\*\* employee in department 30

SELECT first\_name, salary

FROM employees

WHERE salary > ANY (

SELECT salary

FROM employees

WHERE department\_id = 30

);

--all

-- Get employees whose salary is greater than \*\*all\*\* employees in department 30

SELECT first\_name, salary

FROM employees

WHERE salary > ALL (

SELECT salary

FROM employees

WHERE department\_id = 30

);

--exists

-- Get departments that have at least one employee

SELECT department\_id, department\_name

FROM departments d

WHERE EXISTS (

SELECT 1

FROM employees e

WHERE e.department\_id = d.department\_id

);

--not exists

-- Get departments that have no employees

SELECT department\_id, department\_name

FROM departments d

WHERE NOT EXISTS (

SELECT 1

FROM employees e

WHERE e.department\_id = d.department\_id

);

--2A OR 1A

--union

-- Get all department IDs from employees and departments (no duplicates)

SELECT department\_id FROM employees

UNION

SELECT department\_id FROM departments;

--intersect

-- Get department IDs that exist in both employees and departments

SELECT department\_id FROM employees

INTERSECT

SELECT department\_id FROM departments;

--minus

-- Get departments with no employees

SELECT department\_id FROM departments

MINUS

SELECT department\_id FROM employees;

--PRIMARY KEY Uniquely identifies each row (not null)

--FOREIGN KEY Ensures value exists in another table

--NOT NULL Prevents null values

--UNIQUE Ensures all values are unique

--CHECK Ensures values meet a condition

--DEFAULT Sets a default value if none is provided

--1b &7b

SET SERVEROUTPUT ON;

DECLARE

-- Cursor with parameter

CURSOR emp\_cursor (p\_dept\_id employees.department\_id%TYPE) IS

SELECT first\_name, last\_name, salary

FROM employees

WHERE department\_id = p\_dept\_id;

-- Variables to hold employee data

v\_fname employees.first\_name%TYPE;

v\_lname employees.last\_name%TYPE;

v\_salary employees.salary%TYPE;

BEGIN

-- Open the cursor for department 60 (you can change this)

OPEN emp\_cursor(60);

LOOP

FETCH emp\_cursor INTO v\_fname, v\_lname, v\_salary;

EXIT WHEN emp\_cursor%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Name: ' || v\_fname || ' ' || v\_lname || ' | Salary: ' || v\_salary);

END LOOP;

CLOSE emp\_cursor;

END;

/

--2B

DECLARE

salary\_increment NUMBER := 500;

max\_salary\_limit NUMBER := 20000;

high\_salary\_exception EXCEPTION;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Processing employees...');

DECLARE

min\_salary NUMBER := 5000;

BEGIN

WHILE min\_salary <= max\_salary\_limit LOOP

DBMS\_OUTPUT.PUT\_LINE('Checking salaries above: ' || min\_salary);

FOR emp IN (SELECT employee\_id, first\_name, salary

FROM employees

WHERE salary > 5000 AND salary <= 5000 + 5000) LOOP

DBMS\_OUTPUT.PUT\_LINE('Processing Employee: ' || emp.first\_name || ' with Salary: ' || emp.salary);

IF emp.salary + salary\_increment > max\_salary\_limit THEN

RAISE high\_salary\_exception;

ELSE

UPDATE employees

SET salary = salary + salary\_increment

WHERE employee\_id = emp.employee\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated Salary for Employee ID: ' || emp.employee\_id);

END IF;

END LOOP;

min\_salary := min\_salary + 5000;

END LOOP;

END;

DBMS\_OUTPUT.PUT\_LINE('Processing completed.');

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No employees found for the given conditions.');

WHEN high\_salary\_exception THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Salary increment exceeds the allowed limit.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END;

--3a

--Aggregate functions

SELECT COUNT(\*) AS total\_employees

FROM employees;

SELECT SUM(salary) AS total\_salary

FROM employees;

SELECT AVG(salary) AS avg\_salary

FROM employees;

SELECT MAX(salary) AS highest\_salary

FROM employees;

SELECT MIN(salary) AS lowest\_salary

FROM employees;

CREATE TABLE sales (

salesperson VARCHAR2(20),

region VARCHAR2(20),

amount NUMBER(10)

);

INSERT INTO sales VALUES ('John', 'East', 5000);

INSERT INTO sales VALUES ('Alice', 'West', 7000);

INSERT INTO sales VALUES ('John', 'East', 4000);

INSERT INTO sales VALUES ('Alice', 'West', 3000);

INSERT INTO sales VALUES ('Mark', 'East', 3500);

COMMIT;

--GROUP BY --4A

SELECT salesperson, SUM(amount) AS total\_sales

FROM sales

GROUP BY salesperson;

--GROUP BY HAVING

SELECT salesperson, SUM(amount) AS total\_sales

FROM sales

GROUP BY salesperson

HAVING SUM(amount) > 8000;

--Creating and Dropping Views --5A

CREATE OR REPLACE VIEW sales\_summary\_view AS

SELECT salesperson, SUM(amount) AS total\_sales

FROM sales

GROUP BY salesperson;

--3B

CREATE OR REPLACE FUNCTION get\_job\_id (

p\_emp\_id IN employees.employee\_id%TYPE

) RETURN employees.job\_id%TYPE

IS

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

BEGIN

SELECT job\_id INTO v\_job\_id

FROM employees

WHERE employee\_id = p\_emp\_id;

RETURN v\_job\_id;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 'NO\_JOB';

WHEN OTHERS THEN

RETURN 'ERROR';

END;

/

SET SERVEROUTPUT ON;

BEGIN

DBMS\_OUTPUT.PUT\_LINE('Job ID: ' || get\_job\_id(101));

END;

/

--4B

-- Procedure 1: Fetch employee details by EmployeeID

DESC employees;

set serveroutput on;

CREATE OR REPLACE PROCEDURE GetEmployeeDetails (

p\_EmployeeID IN employees.employee\_id%TYPE,

p\_Name OUT VARCHAR2,

) IS

BEGIN

SELECT first\_name || ' ' || last\_name, salary

INTO p\_Name, p\_Salary

FROM employees

WHERE employee\_id = p\_EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || p\_Name);

DBMS\_OUTPUT.PUT\_LINE('Employee Salary: ' || p\_Salary);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No employee found with EmployeeID ' || p\_EmployeeID);

p\_Name := NULL;

p\_Salary := NULL;

WHEN OTHERS THEN

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

p\_Name := NULL;

p\_Salary := NULL;

END GetEmployeeDetails;

/

-- Procedure 2: Update employee salary by EmployeeID

DESC employees;

set serveroutput on;

CREATE OR REPLACE PROCEDURE GetEmployeeDetails (

p\_EmployeeID IN employees.employee\_id%TYPE,

p\_Name OUT VARCHAR2,

p\_Salary OUT employees.salary%TYPE

) IS

BEGIN

SELECT first\_name || ' ' || last\_name, salary

INTO p\_Name, p\_Salary

FROM employees

WHERE employee\_id = p\_EmployeeID;

DBMS\_OUTPUT.PUT\_LINE('Employee Name: ' || p\_Name);

DBMS\_OUTPUT.PUT\_LINE('Employee Salary: ' || p\_Salary);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No employee found with EmployeeID ' || p\_EmployeeID);

p\_Name := NULL;

p\_Salary := NULL;

WHEN OTHERS THEN

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

p\_Name := NULL;

p\_Salary := NULL;

END GetEmployeeDetails;

/

-- Example usage in anonymous block:

set serveroutput on;

DECLARE

v\_Name VARCHAR2(50);

v\_Salary employees.salary%TYPE;

v\_NewSalary employees.salary%TYPE := 55000;

BEGIN

GetEmployeeDetails(p\_EmployeeID => 101, p\_Name => v\_Name, p\_Salary => v\_Salary);

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

DBMS\_OUTPUT.PUT\_LINE('Current Salary: ' || v\_Salary);

UpdateEmployeeSalary(p\_EmployeeID => 101, p\_NewSalary => v\_NewSalary, p\_Salary => v\_Salary);

DBMS\_OUTPUT.PUT\_LINE('Updated Salary: ' || v\_Salary);

END;

/

--5B

DECLARE

-- Variables for employee data

v\_EmployeeID Employees.Employee\_ID%TYPE;

v\_Name Employees.first\_Name%TYPE;

v\_Salary Employees.Salary%TYPE;

v\_Category VARCHAR2(20);

v\_Grade VARCHAR2(20);

BEGIN

select employee\_id,first\_name,salary into

v\_EmployeeID,v\_Name,v\_Salary from employees where employee\_id = 105;

IF v\_Salary IS NOT NULL THEN

IF v\_Salary>= 15000 THEN

v\_Grade := 'High Salary';

ELSIF v\_Salary>= 10000 THEN

v\_Grade := 'Medium Salary';

ELSE

v\_Grade := 'Low Salary';

END IF;

ELSE

v\_Grade := 'Salary not available';

END IF;

-- Using CASE expression to categorize the employee based on salary

v\_Category := CASE

WHEN v\_Salary>= 15000 THEN 'Top Performer'

WHEN v\_Salary>= 10000 THEN 'Average Performer'

ELSE 'Needs Improvement'

END;

-- Handling NULL values with NULLIF and COALESCE

DBMS\_OUTPUT.PUT\_LINE('Employee ID: ' || v\_EmployeeID);

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

DBMS\_OUTPUT.PUT\_LINE('Salary: ' || COALESCE(v\_Salary, 0));

DBMS\_OUTPUT.PUT\_LINE('Grade: ' || v\_Grade);

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

EXCEPTION

WHEN OTHERS THEN

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

END;

--6A

--Conversion Functions

--1.TO\_CHAR: Convert a number or date to a string.

SELECT salary,TO\_CHAR(Salary, '999,999.99') AS FormattedSalary FROM Employees;

--2.TO\_NUMBER: Convert a string to a number.

SELECT TO\_NUMBER('12345.67') AS NumericValue FROM DUAL;

--3.TO\_DATE: Convert a string to a date.

SELECT SYSDATE, TO\_CHAR(SYSDATE, 'DD-MON-YYYY') AS formatted\_date FROM dual;

--6B

SET SERVEROUTPUT ON;

DECLARE

-- Declaration Section

num1 NUMBER := 10;

num2 NUMBER := 0;

result NUMBER;

BEGIN

-- Executable Section

result := num1 / num2;

DBMS\_OUTPUT.PUT\_LINE('Result is: ' || result);

EXCEPTION

-- Exception Handling Section

WHEN ZERO\_DIVIDE THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Division by zero is not allowed.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred: ' || SQLERRM);

END;

/

--7a

select upper('sql course') from dual;

select lower('SQL COURSE') from dual;

select initcap('SQL COURSE') from dual;

select length('SQL COURSE') from dual;

select substr('SQL COURSE',5,6) from dual;

select instr('SQL COURSE','R') from dual;

select replace('SQL Course','SQL','DBMS') from dual;

SELECT TRIM(' SQL Course ')AS TrimmedLeft, RTRIM(' SQL Course ') AS TrimmedRight FROM dual;

select lpad('SQL COURSE',15,'\*') from dual;

select rpad('SQL COURSE',15,'\*') from dual;

select concat('SQl', 'Course')from dual;

--8a

--SYSDATE – Returns the current date and time

SELECT SYSDATE AS today FROM dual;

--NEXT\_DAY(date, 'DAY') – Returns the next specified weekday after the given date

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

--ADD\_MONTHS(date, n) – Adds or subtracts months

SELECT ADD\_MONTHS(SYSDATE, 3) FROM dual;

--MONTHS\_BETWEEN(date1, date2) – Returns number of months between two dates

SELECT MONTHS\_BETWEEN(SYSDATE, TO\_DATE('01-JAN-2024', 'DD-MON-YYYY')) AS months\_gap

FROM dual;

--LEAST(date1, date2, ...) – Returns the earliest date

SELECT LEAST(SYSDATE, TO\_DATE('01-JAN-2024', 'DD-MON-YYYY')) AS earliest\_date

FROM dual;

--REATEST(date1, date2, ...) – Returns the latest date

SELECT GREATEST(SYSDATE, TO\_DATE('01-JAN-2024', 'DD-MON-YYYY')) AS latest\_date

FROM dual;

--TRUNC(date) – Truncates time part (sets time to 00:00:00)

SELECT TRUNC(SYSDATE) AS date\_only FROM dual;

--ROUND(date) – Rounds to the nearest day

SELECT ROUND(SYSDATE) AS rounded\_date FROM dual;

--TO\_CHAR(date, format) – Converts date to string

SELECT TO\_CHAR(SYSDATE, 'DD-MON-YYYY HH24:MI:SS') AS formatted\_now FROM dual;

--TO\_DATE(string, format) – Converts string to date

SELECT TO\_DATE('09-APR-2025', 'DD-MON-YYYY') AS actual\_date FROM dual;

--8b

CREATE OR REPLACE FUNCTION get\_annual\_salary(emp\_id IN NUMBER)

RETURN NUMBER

IS

v\_salary employees.salary%TYPE;

v\_annual\_salary NUMBER;

BEGIN

-- Get monthly salary

SELECT salary INTO v\_salary

FROM employees

WHERE employee\_id = emp\_id;

-- Calculate annual salary

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

RETURN v\_annual\_salary;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

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

RETURN NULL;

WHEN OTHERS THEN

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

RETURN NULL;

END;

/

SELECT get\_annual\_salary(101) AS annual\_salary FROM dual;

SELECT

employee\_id,

first\_name,

salary,

get\_annual\_salary(employee\_id) AS annual\_salary

FROM employees

WHERE department\_id = 60;

SET SERVEROUTPUT ON;

DECLARE

v\_result NUMBER;

BEGIN

v\_result := get\_annual\_salary(101);

DBMS\_OUTPUT.PUT\_LINE('Annual Salary: ' || v\_result);

END;

/