**2.Consider the following schema for a Hospital Database: DOCTOR (Did, Dname, DAddress, Qualification) PATIENTMASTER (Pcode, Pname, Padd, age, gender, bloodgroup, Pid) ADMMITTEDPATIENT(Pcode, EntryDate, DischargeDate, WardNo, Disease)**

**a) Find the detail of the doctor who is treating the patient of ward no3**

**b) Find the detail of patient who are admitted within period 03/03/2020 to 25/05/2020**

**c) Find the detail of patient who are suffered from blood cancer**

**d) Create view on DOCTOR & PATIENTASTER tables.**

**a) SELECT DOCTOR.Did, DOCTOR.Dname, DOCTOR.DAddress, DOCTOR.Qualification**

**FROM DOCTOR**

**JOIN ADMMITTEDPATIENT ON DOCTOR.Did = ADMMITTEDPATIENT.Did**

**WHERE ADMMITTEDPATIENT.WardNo = 3;**

**b)** **SELECT P.Pcode, P.Pname, P.Padd, P.age, P.gender, P.bloodgroup, P.Pid**

**FROM PATIENTMASTER P**

**JOIN ADMMITTEDPATIENT AP ON P.Pcode = AP.Pcode**

**WHERE AP.EntryDate BETWEEN '2020-03-03' AND '2020-05-25';**

**c) SELECT P.Pcode, P.Pname, P.Padd, P.age, P.gender, P.bloodgroup, P.Pid**

**FROM PATIENTMASTER P**

**JOIN ADMMITTEDPATIENT AP ON P.Pcode = AP.Pcode**

**WHERE AP.Disease = 'Blood Cancer';**

**d) CREATE VIEW DoctorPatientView AS**

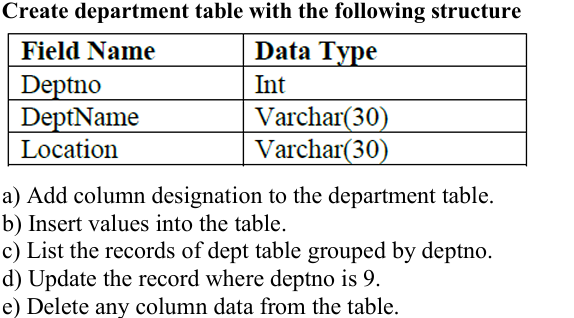
**SELECT D.Did, D.Dname, D.DAddress, D.Qualification,**

**P.Pcode, P.Pname, P.Padd, P.age, P.gender, P.bloodgroup, P.Pid**

**FROM DOCTOR D**

**JOIN PATIENTMASTER P ON D.Did = P.Pid;**

**3.**

****

1. **CREATE TABLE DEPARTMENT ( DeptNo INT PRIMARY KEY, -- Department number (Primary Key) DeptName VARCHAR(50), -- Department name Location VARCHAR(100) -- Location of the department );**
2. **ALTER TABLE DEPARTMENT ADD Designation VARCHAR(50);**
3. **INSERT INTO DEPARTMENT (DeptNo, DeptName, Location, Designation) VALUES (1, 'HR', 'New York', 'Manager'), (2, 'Finance', 'London', 'Accountant'), (3, 'IT', 'San Francisco', 'Developer'), (4, 'Sales', 'Los Angeles', 'Sales Executive');**
4. **SELECT DeptNo, DeptName, Location, Designation FROM DEPARTMENT GROUP BY DeptNo, DeptName, Location, Designation**;
5. **UPDATE DEPARTMENT SET Location = 'Chicago', Designation = 'Senior Manager' WHERE DeptNo = 9;**
6. **UPDATE DEPARTMENT SET Designation = NULL WHERE DeptNo = 2;**
7. **ALTER TABLE DEPARTMENT DROP COLUMN Designation;**

**4.Create database using following schema. Apply Integrity Constraints and answer the following queries using SQL. DOCTOR (Did, Dname, Daddress, qualification) PATIENT (Pid, Pname, age, gender) Integrity Constraints:**

**1. The values of any attributes should not be null.**

**2. Did should be unique constraints.**

**3. Pid should be unique constraints.**

**4. Gender value should be M (male) or F (female).**

**Queries: a) Insert at least 10 records in table.**

**b) Find details of all doctors.**

**c) Delete the records from DOCTOR where qualification is M.S**

**d) Find details of patient where age is less than 40.**

**e) Update the patient name where patient id is 5.**

**1.CREATE TABLE DOCTOR ( Did INT PRIMARY KEY, Dname VARCHAR(100) NOT NULL, Daddress VARCHAR(255) NOT NULL, qualification VARCHAR(50) NOT NULL, CONSTRAINT unique\_Did UNIQUE (Did) );**

**CREATE TABLE PATIENT ( Pid INT PRIMARY KEY, Pname VARCHAR(100) NOT NULL, age INT NOT NULL, gender CHAR(1) CHECK (gender IN ('M', 'F')), CONSTRAINT unique\_Pid UNIQUE (Pid) );**

INSERT INTO DOCTOR (Did, Dname, Daddress, qualification)

VALUES

(1, 'Dr. Smith', '123 Elm St, City', 'M.D.'),

(2, 'Dr. Johnson', '456 Oak St, City', 'M.S'),

(3, 'Dr. Lee', '789 Pine St, City', 'PhD'),

(4, 'Dr. Patel', '321 Birch St, City', 'MBBS'),

(5, 'Dr. Davis', '654 Maple St, City', 'M.D.'),

(6, 'Dr. Williams', '987 Cedar St, City', 'M.S'),

(7, 'Dr. Brown', '135 Birch St, City', 'D.M.S'),

(8, 'Dr. Taylor', '246 Pine St, City', 'MBBS'),

(9, 'Dr. Moore', '357 Oak St, City', 'M.D.'),

(10, 'Dr. White', '468 Elm St, City', 'M.S');

-- Inserting records into PATIENT table

INSERT INTO PATIENT (Pid, Pname, age, gender)

VALUES

(1, 'Alice', 29, 'F'),

(2, 'Bob', 45, 'M'),

(3, 'Charlie', 32, 'M'),

(4, 'Diana', 50, 'F'),

(5, 'Eve', 22, 'F'),

(6, 'Frank', 60, 'M'),

(7, 'Grace', 25, 'F'),

(8, 'Hank', 39, 'M'),

(9, 'Ivy', 33, 'F'),

(10, 'Jack', 41, 'M');

**3. Queries based on the given requirements:**

**b) Find details of all doctors:**

sql

Copy

SELECT \* FROM DOCTOR;

**c) Delete the records from DOCTOR where qualification is 'M.S':**

sql

Copy

DELETE FROM DOCTOR WHERE qualification = 'M.S';

**d) Find details of patients where age is less than 40:**

sql

Copy

SELECT \* FROM PATIENT WHERE age < 40;

**e) Update the patient name where patient id is 5:**

sql

Copy

UPDATE PATIENT

SET Pname = 'Eva'

WHERE Pid = 5;

**Notes:**

* The integrity constraints ensure that there are no NULL values in the Did, Dname, Daddress, qualification, Pid, Pname, age, and gender fields.
* The Did and Pid fields have unique constraints, ensuring no duplicates.
* The gender column allows only values 'M' or 'F' for male and female respectively.

Let me know if you need further explanations or adjustments!

**5.Write a PL/SQL code to create an employee database with the tables and fields specified as below. Employee[Emp no Employee name Street City] Works[Emp no Company\_name Joining\_date Designation Salary] Company[Emp no City] Manages[Emp no Manager\_name,Mang\_no]**

**CREATE TABLE Employee (**

**emp\_no NUMBER PRIMARY KEY,**

**emp\_name VARCHAR2(100),**

**street VARCHAR2(100),**

**city VARCHAR2(50)**

**);**

**CREATE TABLE Works (**

**emp\_no NUMBER,**

**company\_name VARCHAR2(100),**

**joining\_date DATE,**

**designation VARCHAR2(50),**

**salary NUMBER(10, 2),**

**PRIMARY KEY (emp\_no, company\_name),**

**FOREIGN KEY (emp\_no) REFERENCES Employee(emp\_no)**

**);**

**CREATE TABLE Company (**

**emp\_no NUMBER,**

**city VARCHAR2(50),**

**PRIMARY KEY (emp\_no),**

**FOREIGN KEY (emp\_no) REFERENCES Employee(emp\_no)**

**);**

**CREATE TABLE Manages (**

**emp\_no NUMBER,**

**manager\_name VARCHAR2(100),**

**mang\_no NUMBER,**

**PRIMARY KEY (emp\_no),**

**FOREIGN KEY (emp\_no) REFERENCES Employee(emp\_no)**

**);**

**INSERT INTO Employee (emp\_no, emp\_name, street, city)**

**VALUES (1, 'John Doe', '123 Elm St', 'New York');**

**INSERT INTO Employee (emp\_no, emp\_name, street, city)**

**VALUES (2, 'Jane Smith', '456 Oak St', 'Chicago');**

**INSERT INTO Works (emp\_no, company\_name, joining\_date, designation, salary)**

**VALUES (1, 'ABC Corp', TO\_DATE('2023-01-01', 'YYYY-MM-DD'), 'Software Engineer', 85000);**

**INSERT INTO Works (emp\_no, company\_name, joining\_date, designation, salary)**

**VALUES (2, 'XYZ Ltd', TO\_DATE('2022-06-15', 'YYYY-MM-DD'), 'Data Analyst', 75000);**

**INSERT INTO Company (emp\_no, city)**

**VALUES (1, 'New York');**

**INSERT INTO Company (emp\_no, city)**

**VALUES (2, 'Chicago');**

**INSERT INTO Manages (emp\_no, manager\_name, mang\_no)**

**VALUES (1, 'Sarah Johnson', 101);**

**INSERT INTO Manages (emp\_no, manager\_name, mang\_no)**

**VALUES (2, 'Michael Brown', 102);**

**6.Write a PL/SQL code to create an employee database with the tables and fields specified as below. Employee[Emp no Employee name Street City] Works[Emp no Company\_name Joining\_date Designation Salary] Company[Emp no City] Manages[Emp no Manager\_name,Mang\_no]**

**CREATE TABLE Employee (**

**Emp\_no NUMBER PRIMARY KEY,**

**Employee\_name VARCHAR2(100),**

**Street VARCHAR2(100),**

**City VARCHAR2(50)**

**);**

**CREATE TABLE Works (**

**Emp\_no NUMBER,**

**Company\_name VARCHAR2(100),**

**Joining\_date DATE,**

**Designation VARCHAR2(100),**

**Salary NUMBER(10,2),**

**CONSTRAINT fk\_works\_emp FOREIGN KEY (Emp\_no) REFERENCES Employee(Emp\_no)**

**);**

**CREATE TABLE Company (**

**Emp\_no NUMBER,**

**City VARCHAR2(50),**

**CONSTRAINT fk\_company\_emp FOREIGN KEY (Emp\_no) REFERENCES Employee(Emp\_no)**

**);**

**CREATE TABLE Manages (**

**Emp\_no NUMBER,**

**Manager\_name VARCHAR2(100),**

**Mang\_no NUMBER,**

**CONSTRAINT fk\_manages\_emp FOREIGN KEY (Emp\_no) REFERENCES Employee(Emp\_no),**

**CONSTRAINT fk\_manages\_mang FOREIGN KEY (Mang\_no) REFERENCES Employee(Emp\_no)**

**);**

**6. PL/SQL code to retrieve the employee name, join date, and designation from employee database of an employee whose number is input by the user.**

**CREATE OR REPLACE PROCEDURE get\_employee\_details(**

**emp\_number IN NUMBER,**

**emp\_name OUT VARCHAR2,**

**join\_date OUT DATE,**

**designation OUT VARCHAR2**

**)**

**IS**

**BEGIN**

**-- Fetch employee details based on the given employee number**

**SELECT name, hire\_date, job\_title**

**INTO emp\_name, join\_date, designation**

**FROM employees**

**WHERE emp\_id = emp\_number;**

**EXCEPTION**

**-- Handle case when employee number is not found**

**WHEN NO\_DATA\_FOUND THEN**

**emp\_name := 'Not Found';**

**join\_date := NULL;**

**designation := 'Not Found';**

**-- Handle any other errors**

**WHEN OTHERS THEN**

**emp\_name := 'Error';**

**join\_date := NULL;**

**designation := 'Error';**

**END get\_employee\_details;**

**/**

**7.select \* from hr.employees**

**create table employees as select \* from hr.employees;**

**DECLARE**

**CURSOR emp\_cursor IS**

**SELECT EMPLOYEE\_ID, salary**

**FROM employees**

**WHERE salary < (SELECT AVG(salary) FROM employees);**

**v\_emp\_id employees.EMPLOYEE\_ID%TYPE;**

**v\_salary employees.salary%TYPE;**

**v\_increment NUMBER := 500; -- Salary increment amount**

**BEGIN**

**OPEN emp\_cursor;**

**LOOP**

**FETCH emp\_cursor INTO v\_emp\_id, v\_salary;**

**EXIT WHEN emp\_cursor%NOTFOUND;**

**-- Update salary**

**UPDATE employees**

**SET salary = salary + v\_increment**

**WHERE EMPLOYEE\_ID = v\_emp\_id;**

**COMMIT; -- Commit after each update to avoid rollback issues in Oracle 10g**

**END LOOP;**

**CLOSE emp\_cursor;**

**DBMS\_OUTPUT.PUT\_LINE('Salaries updated successfully.');**

**EXCEPTION**

**WHEN OTHERS THEN**

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

**ROLLBACK; -- Undo changes in case of error**

**END;**

**select \* from employees**

**where employee\_id =119**

**13**

**-- Package Specification**

**CREATE OR REPLACE PACKAGE student\_pkg AS**

**PROCEDURE insert\_student(p\_id NUMBER, p\_name VARCHAR2, p\_age NUMBER, p\_course VARCHAR2);**

**PROCEDURE update\_student(p\_id NUMBER, p\_name VARCHAR2, p\_age NUMBER, p\_course VARCHAR2);**

**PROCEDURE delete\_student(p\_id NUMBER);**

**PROCEDURE get\_student(p\_id NUMBER);**

**END student\_pkg;**

**/**

**-- Package Body**

**CREATE OR REPLACE PACKAGE BODY student\_pkg AS**

**PROCEDURE insert\_student(p\_id NUMBER, p\_name VARCHAR2, p\_age NUMBER, p\_course VARCHAR2) IS**

**BEGIN**

**INSERT INTO student (id, name, age, course)**

**VALUES (p\_id, p\_name, p\_age, p\_course);**

**COMMIT;**

**END insert\_student;**

**PROCEDURE update\_student(p\_id NUMBER, p\_name VARCHAR2, p\_age NUMBER, p\_course VARCHAR2) IS**

**BEGIN**

**UPDATE student**

**SET name = p\_name, age = p\_age, course = p\_course**

**WHERE id = p\_id;**

**COMMIT;**

**END update\_student;**

**PROCEDURE delete\_student(p\_id NUMBER) IS**

**BEGIN**

**DELETE FROM student WHERE id = p\_id;**

**COMMIT;**

**END delete\_student;**

**PROCEDURE get\_student(p\_id NUMBER) IS**

**v\_name student.name%TYPE;**

**v\_age student.age%TYPE;**

**v\_course student.course%TYPE;**

**BEGIN**

**SELECT name, age, course INTO v\_name, v\_age, v\_course**

**FROM student WHERE id = p\_id;**

**DBMS\_OUTPUT.PUT\_LINE('ID: ' || p\_id || ', Name: ' || v\_name || ', Age: ' || v\_age || ', Course: ' || v\_course);**

**END get\_student;**

**END student\_pkg;**

**/**

**14.**

**DECLARE**

**-- User-defined exception**

**negative\_value EXCEPTION;**

**PRAGMA EXCEPTION\_INIT(negative\_value, -20001);**

**-- Built-in exception (for division by zero)**

**v\_divisor NUMBER := 0;**

**-- Variable to test exceptions**

**v\_value NUMBER := -5;**

**BEGIN**

**-- Handling built-in exception (ZeroDivisionError equivalent)**

**IF v\_divisor = 0 THEN**

**RAISE ZERO\_DIVIDE;**

**END IF;**

**-- Handling user-defined exception**

**IF v\_value < 0 THEN**

**RAISE negative\_value;**

**END IF;**

**DBMS\_OUTPUT.PUT\_LINE('Valid input received: ' || v\_value);**

**EXCEPTION**

**WHEN ZERO\_DIVIDE THEN**

**DBMS\_OUTPUT.PUT\_LINE('Caught a built-in exception: Division by zero is not allowed.');**

**WHEN negative\_value THEN**

**DBMS\_OUTPUT.PUT\_LINE('Caught a user-defined exception: Negative values are not allowed.');**

**WHEN OTHERS THEN**

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

**END;**

**15.**

**DECLARE**

**v\_input\_string VARCHAR2(100) := 'PLSQLExample'; -- Input string**

**v\_reversed\_string VARCHAR2(100) := '';**

**v\_length NUMBER;**

**BEGIN**

**-- Get the length of the input string**

**v\_length := LENGTH(v\_input\_string);**

**-- Loop through the string in reverse order**

**FOR i IN REVERSE 1..v\_length LOOP**

**v\_reversed\_string := v\_reversed\_string || SUBSTR(v\_input\_string, i, 1);**

**END LOOP;**

**-- Output the reversed string**

**DBMS\_OUTPUT.PUT\_LINE('Original String: ' || v\_input\_string);**

**DBMS\_OUTPUT.PUT\_LINE('Reversed String: ' || v\_reversed\_string);**

**END;**

**16.**

**-- First, create the audit table to store changes**

**CREATE TABLE EMPLOYEES\_AUDIT (**

**AUDIT\_ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,**

**EMPLOYEE\_ID NUMBER,**

**OPERATION\_TYPE VARCHAR2(10),**

**OPERATION\_TIMESTAMP TIMESTAMP DEFAULT SYSTIMESTAMP**

**);**

**-- Now, create the trigger**

**CREATE OR REPLACE TRIGGER employees\_audit\_trigger**

**AFTER INSERT OR UPDATE OR DELETE ON EMPLOYEES**

**FOR EACH ROW**

**BEGIN**

**IF INSERTING THEN**

**INSERT INTO EMPLOYEES\_AUDIT (EMPLOYEE\_ID, OPERATION\_TYPE, OPERATION\_TIMESTAMP)**

**VALUES (:NEW.EMPLOYEE\_ID, 'INSERT', SYSTIMESTAMP);**

**ELSIF UPDATING THEN**

**INSERT INTO EMPLOYEES\_AUDIT (EMPLOYEE\_ID, OPERATION\_TYPE, OPERATION\_TIMESTAMP)**

**VALUES (:NEW.EMPLOYEE\_ID, 'UPDATE', SYSTIMESTAMP);**

**ELSIF DELETING THEN**

**INSERT INTO EMPLOYEES\_AUDIT (EMPLOYEE\_ID, OPERATION\_TYPE, OPERATION\_TIMESTAMP)**

**VALUES (:OLD.EMPLOYEE\_ID, 'DELETE', SYSTIMESTAMP);**

**END IF;**

**END;**

**/**

**17.**

**-- First, create the audit table to store changes**

**CREATE TABLE EMPLOYEES\_AUDIT (**

**AUDIT\_ID NUMBER GENERATED ALWAYS AS IDENTITY PRIMARY KEY,**

**EMPLOYEE\_ID NUMBER,**

**OPERATION\_TYPE VARCHAR2(10),**

**OPERATION\_TIMESTAMP TIMESTAMP DEFAULT SYSTIMESTAMP**

**);**

**-- Now, create the trigger**

**CREATE OR REPLACE TRIGGER employees\_audit\_trigger**

**AFTER INSERT OR UPDATE OR DELETE ON EMPLOYEES**

**FOR EACH ROW**

**BEGIN**

**IF INSERTING THEN**

**INSERT INTO EMPLOYEES\_AUDIT (EMPLOYEE\_ID, OPERATION\_TYPE, OPERATION\_TIMESTAMP)**

**VALUES (:NEW.EMPLOYEE\_ID, 'INSERT', SYSTIMESTAMP);**

**ELSIF UPDATING THEN**

**INSERT INTO EMPLOYEES\_AUDIT (EMPLOYEE\_ID, OPERATION\_TYPE, OPERATION\_TIMESTAMP)**

**VALUES (:NEW.EMPLOYEE\_ID, 'UPDATE', SYSTIMESTAMP);**

**ELSIF DELETING THEN**

**INSERT INTO EMPLOYEES\_AUDIT (EMPLOYEE\_ID, OPERATION\_TYPE, OPERATION\_TIMESTAMP)**

**VALUES (:OLD.EMPLOYEE\_ID, 'DELETE', SYSTIMESTAMP);**

**END IF;**

**END;**

**/**

**-- PL/SQL program to calculate a 10% bonus for employees with salary > 50,000**

**DECLARE**

**CURSOR emp\_cursor IS**

**SELECT EMPLOYEE\_ID, NAME, SALARY FROM EMPLOYEES WHERE SALARY > 50000;**

**v\_emp\_id EMPLOYEES.EMPLOYEE\_ID%TYPE;**

**v\_name EMPLOYEES.NAME%TYPE;**

**v\_salary EMPLOYEES.SALARY%TYPE;**

**v\_bonus NUMBER;**

**BEGIN**

**OPEN emp\_cursor;**

**LOOP**

**FETCH emp\_cursor INTO v\_emp\_id, v\_name, v\_salary;**

**EXIT WHEN emp\_cursor%NOTFOUND;**

**v\_bonus := v\_salary \* 0.10;**

**DBMS\_OUTPUT.PUT\_LINE('Employee: ' || v\_name || ', Bonus: ' || v\_bonus);**

**END LOOP;**

**CLOSE emp\_cursor;**

**END;**

**/**

**18.**

**SELECT**

**COUNT(\*) AS TOTAL\_EMPLOYEES,**

**AVG(SALARY) AS AVERAGE\_SALARY,**

**MAX(SALARY) AS HIGHEST\_SALARY,**

**MIN(SALARY) AS LOWEST\_SALARY,**

**SUM(SALARY) AS TOTAL\_SALARY**

**FROM EMPLOYEES;**

**19.**

**DECLARE**

**v\_start NUMBER := 1; -- Define the start of the range**

**v\_end NUMBER := 20; -- Define the end of the range**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE('Even Numbers between ' || v\_start || ' and ' || v\_end || ':');**

**FOR i IN v\_start..v\_end LOOP**

**IF MOD(i, 2) = 0 THEN**

**DBMS\_OUTPUT.PUT\_LINE(i);**

**END IF;**

**END LOOP;**

**END;**

**20.**

**DECLARE**

**num1 NUMBER := 10; -- First number**

**num2 NUMBER := 25; -- Second number**

**num3 NUMBER := 15; -- Third number**

**largest NUMBER;**

**BEGIN**

**-- Determine the largest number**

**IF num1 >= num2 AND num1 >= num3 THEN**

**largest := num1;**

**ELSIF num2 >= num1 AND num2 >= num3 THEN**

**largest := num2;**

**ELSE**

**largest := num3;**

**END IF;**

**DBMS\_OUTPUT.PUT\_LINE('The largest number is: ' || largest);**

**END;.**

**10.**

**CREATE OR REPLACE PROCEDURE Get\_Student\_Performance(p\_course\_id IN NUMBER) IS**

**v\_100\_70 NUMBER := 0;**

**v\_69\_60 NUMBER := 0;**

**v\_59\_50 NUMBER := 0;**

**v\_below\_49 NUMBER := 0;**

**BEGIN**

**-- Count students in each range**

**SELECT**

**SUM(CASE WHEN percentage BETWEEN 70 AND 100 THEN 1 ELSE 0 END),**

**SUM(CASE WHEN percentage BETWEEN 60 AND 69 THEN 1 ELSE 0 END),**

**SUM(CASE WHEN percentage BETWEEN 50 AND 59 THEN 1 ELSE 0 END),**

**SUM(CASE WHEN percentage < 50 THEN 1 ELSE 0 END)**

**INTO v\_100\_70, v\_69\_60, v\_59\_50, v\_below\_49**

**FROM student\_course**

**WHERE course\_id = p\_course\_id;**

**-- Display results**

**DBMS\_OUTPUT.PUT\_LINE('Course ID: ' || p\_course\_id);**

**DBMS\_OUTPUT.PUT\_LINE('Students scoring 100-70%: ' || v\_100\_70);**

**DBMS\_OUTPUT.PUT\_LINE('Students scoring 69-60%: ' || v\_69\_60);**

**DBMS\_OUTPUT.PUT\_LINE('Students scoring 59-50%: ' || v\_59\_50);**

**DBMS\_OUTPUT.PUT\_LINE('Students scoring below 49%: ' || v\_below\_49);**

**END Get\_Student\_Performance;**

**11.**

**CREATE OR REPLACE FUNCTION Add\_Numbers(p\_num1 IN NUMBER, p\_num2 IN NUMBER) RETURN NUMBER IS**

**v\_sum NUMBER;**

**BEGIN**

**v\_sum := p\_num1 + p\_num2;**

**RETURN v\_sum;**

**END Add\_Numbers;**

**/**

**-- Calling the function**

**DECLARE**

**v\_result NUMBER;**

**BEGIN**

**v\_result := Add\_Numbers(10, 20);**

**DBMS\_OUTPUT.PUT\_LINE('The sum is: ' || v\_result);**

**END;**

**12.**

**CREATE OR REPLACE FUNCTION Get\_Total\_Salary(p\_dept\_id IN NUMBER) RETURN NUMBER IS**

**v\_total\_salary NUMBER := 0;**

**BEGIN**

**-- Calculate total salary for the department**

**SELECT SUM(salary)**

**INTO v\_total\_salary**

**FROM employees**

**WHERE department\_id = p\_dept\_id;**

**RETURN v\_total\_salary;**

**END Get\_Total\_Salary;**

**/**

**-- Calling the function**

**DECLARE**

**v\_dept\_salary NUMBER;**

**BEGIN**

**v\_dept\_salary := Get\_Total\_Salary(10);**

**DBMS\_OUTPUT.PUT\_LINE('Total salary for department 10: ' || v\_dept\_salary);**

**END;**

**8.**

**CREATE OR REPLACE TRIGGER Log\_Salary\_Changes**

**BEFORE UPDATE ON salary**

**FOR EACH ROW**

**BEGIN**

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

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

**END;**

**/**

**9.**

**CREATE OR REPLACE TRIGGER Show\_Old\_New\_Ename**

**BEFORE UPDATE OF ename ON employees**

**FOR EACH ROW**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE('Old Name: ' || :OLD.ename);**

**DBMS\_OUTPUT.PUT\_LINE('New Name: ' || :NEW.ename);**

**END;**

**/**