P1. Write a PL/SQL code block to calculate the area of a circle for a value of radius varying from 5 to 9. Store the radius and the corresponding values of calculated area in an empty table named areas, consisting of two columns, radius and area.

-- Create the areas table if it does not exist

BEGIN

EXECUTE IMMEDIATE 'CREATE TABLE areas (

radius NUMBER,

area NUMBER

)';

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE = -955 THEN

NULL; -- Table already exists, ignore error

ELSE

RAISE;

END IF;

END;

/

-- PL/SQL block to calculate and insert area values

DECLARE

radius\_val NUMBER;

area\_val NUMBER;

BEGIN

-- Loop through radius values from 5 to 9

FOR radius\_val IN 5..9 LOOP

-- Calculate the area of the circle

area\_val := 3.14159 \* radius\_val \* radius\_val;

-- Insert the radius and area into the areas table

INSERT INTO areas (radius, area) VALUES (radius\_val, area\_val);

END LOOP;

-- Commit the transaction to save the data

COMMIT;

END;

/

P2. Write an Unnamed PL/SQL of code for the following requirements: - Schema: Borrower (Rollin, Name, DateofIssue, NameofBook, Status) Fine (Roll\_no,Date,Amt) Accept roll\_no & name of book from user. Check the number of days (from date of issue). 1. If days are between 15 to 30 then fine amounts will be Rs 5 per day. 2. If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day. 3. After submitting the book, status will change from I to R. 4. If condition of fine is true, then details will be stored into fine table.

DECLARE

v\_roll\_no Borrower.Rollin%TYPE;

v\_nameofbook Borrower.NameofBook%TYPE;

v\_dateofissue Borrower.DateofIssue%TYPE;

v\_status Borrower.Status%TYPE;

v\_fine\_amt NUMBER := 0;

v\_days NUMBER;

BEGIN

-- Accept input for roll number and name of book

v\_roll\_no := &roll\_no;

v\_nameofbook := '&name\_of\_book';

-- Retrieve DateofIssue and Status for the given roll number and book name

SELECT DateofIssue, Status

INTO v\_dateofissue, v\_status

FROM Borrower

WHERE Rollin = v\_roll\_no AND NameofBook = v\_nameofbook;

-- Calculate the number of days since the book was issued

v\_days := TRUNC(SYSDATE - v\_dateofissue);

-- Calculate the fine based on the number of days

IF v\_days > 30 THEN

-- Fine calculation for more than 30 days

v\_fine\_amt := (30 \* 5) + ((v\_days - 30) \* 50);

ELSIF v\_days BETWEEN 15 AND 30 THEN

-- Fine calculation for 15 to 30 days

v\_fine\_amt := v\_days \* 5;

END IF;

-- Update the status in the Borrower table if the book is being returned

IF v\_status = 'I' THEN

UPDATE Borrower

SET Status = 'R'

WHERE Rollin = v\_roll\_no AND NameofBook = v\_nameofbook;

END IF;

-- Insert a record into the Fine table if there is a fine

IF v\_fine\_amt > 0 THEN

INSERT INTO Fine (Roll\_no, Date, Amt)

VALUES (v\_roll\_no, SYSDATE, v\_fine\_amt);

END IF;

-- Commit the transaction

COMMIT;

-- Output the result

DBMS\_OUTPUT.PUT\_LINE('Days overdue: ' || v\_days);

DBMS\_OUTPUT.PUT\_LINE('Fine amount: ' || v\_fine\_amt);

DBMS\_OUTPUT.PUT\_LINE('Status updated to Returned (R) in Borrower table.');

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No record found for the given roll number and book name.');

WHEN OTHERS THEN

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

END;

/

P3. Write a PL/SQL block of code using Cursor that will merge the data available in the newly created table N\_Roll Call with the data available in the table O\_RollCall. If the data in the first table already exist in the second table, then that data should be skipped.

DECLARE

CURSOR n\_rollcall\_cursor IS

SELECT roll\_no, name, date FROM N\_RollCall;

v\_roll\_no N\_RollCall.roll\_no%TYPE;

v\_name N\_RollCall.name%TYPE;

v\_date N\_RollCall.date%TYPE;

BEGIN

FOR record IN n\_rollcall\_cursor LOOP

-- Attempt to insert only if roll\_no doesn't exist in O\_RollCall

BEGIN

INSERT INTO O\_RollCall (roll\_no, name, date)

SELECT record.roll\_no, record.name, record.date

FROM dual

WHERE NOT EXISTS (

SELECT 1 FROM O\_RollCall WHERE roll\_no = record.roll\_no

);

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

NULL; -- Skip duplicates

END;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Merge completed successfully.');

END;

/

P4.Write a PL/SQL block for following requirements and handle the exceptions. Roll no. of students will be entered by the user. Attendance of roll no. entered by user will be checked in the Stud table. If attendance is less than 75% then display the message “Term not granted” and set the status in stud table as “Detained”. Otherwise display message “Term granted” and set the status in stud table as “Not Detained”. Student (Roll, Name, Attendance, Status)

DECLARE

v\_roll Stud.Roll%TYPE; -- Variable for roll number

v\_attendance Stud.Attendance%TYPE; -- Variable for attendance

v\_status VARCHAR2(20); -- Variable for status message

BEGIN

-- Prompt the user to enter the roll number

v\_roll := &Enter\_Roll\_No;

-- Fetch attendance for the given roll number

SELECT Attendance INTO v\_attendance

FROM Stud

WHERE Roll = v\_roll;

-- Check attendance and update status

IF v\_attendance < 75 THEN

v\_status := 'Detained';

UPDATE Stud

SET Status = v\_status

WHERE Roll = v\_roll;

DBMS\_OUTPUT.PUT\_LINE('Term not granted');

ELSE

v\_status := 'Not Detained';

UPDATE Stud

SET Status = v\_status

WHERE Roll = v\_roll;

DBMS\_OUTPUT.PUT\_LINE('Term granted');

END IF;

COMMIT; -- Commit the transaction

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No student found with Roll No. ' || v\_roll);

WHEN OTHERS THEN

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

END;

/

CREATE TABLE Stud (

Roll NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Attendance NUMBER(5,2), -- Store attendance as a percentage (e.g., 80.50)

Status VARCHAR2(20) -- Store status as "Detained" or "Not Detained"

);

INSERT INTO Stud (Roll, Name, Attendance, Status) VALUES (1, 'John Doe', 80, NULL);

INSERT INTO Stud (Roll, Name, Attendance, Status) VALUES (2, 'Jane Smith', 70, NULL);

INSERT INTO Stud (Roll, Name, Attendance, Status) VALUES (3, 'Mike Johnson', 65, NULL);

INSERT INTO Stud (Roll, Name, Attendance, Status) VALUES (4, 'Emily Davis', 85, NULL);

COMMIT;

P5.Write a PL/SQL Block to increase the salary of employees by 10% of existing salary, who are having salary less than average salary of organization, whenever such salary updates take place, a record for same is maintained in the increment\_salary table. emp(emp\_no, salary) increment\_salary(emp\_no, salary)

CREATE TABLE emp (

emp\_no NUMBER PRIMARY KEY,

salary NUMBER(10,2)

);

CREATE TABLE increment\_salary (

emp\_no NUMBER,

salary NUMBER(10,2),

increment\_date DATE DEFAULT SYSDATE

);

INSERT INTO emp (emp\_no, salary) VALUES (1, 30000);

INSERT INTO emp (emp\_no, salary) VALUES (2, 40000);

INSERT INTO emp (emp\_no, salary) VALUES (3, 25000);

INSERT INTO emp (emp\_no, salary) VALUES (4, 45000);

COMMIT;

DECLARE

v\_avg\_salary NUMBER(10,2); -- Variable to store the average salary

BEGIN

-- Calculate the average salary of all employees

SELECT AVG(salary) INTO v\_avg\_salary FROM emp;

-- Update salaries and log increments for employees earning below the average salary

FOR employee IN (SELECT emp\_no, salary FROM emp WHERE salary < v\_avg\_salary) LOOP

-- Increase salary by 10%

UPDATE emp

SET salary = salary \* 1.1

WHERE emp\_no = employee.emp\_no;

-- Insert record of salary increment in increment\_salary table

INSERT INTO increment\_salary (emp\_no, salary)

VALUES (employee.emp\_no, employee.salary \* 1.1);

END LOOP;

COMMIT; -- Commit all updates and inserts

DBMS\_OUTPUT.PUT\_LINE('Salary updates completed successfully.');

EXCEPTION

WHEN OTHERS THEN

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

END;

/

P6.Write a Stored Procedure namely proc\_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and 900 categories is first class, if marks 899 and 825 category is Higher Second Class. Write a PL/SQL block for using procedure created with above requirement. Stud\_Marks(name, total\_marks), Result (Roll,Name, Class)

CREATE TABLE Stud\_Marks (

name VARCHAR2(50),

total\_marks NUMBER

);

CREATE TABLE Result (

Roll NUMBER PRIMARY KEY,

name VARCHAR2(50),

Class VARCHAR2(20)

);

INSERT INTO Stud\_Marks (name, total\_marks) VALUES ('John Doe', 1450);

INSERT INTO Stud\_Marks (name, total\_marks) VALUES ('Jane Smith', 960);

INSERT INTO Stud\_Marks (name, total\_marks) VALUES ('Mike Johnson', 880);

INSERT INTO Stud\_Marks (name, total\_marks) VALUES ('Emily Davis', 850);

COMMIT;

CREATE OR REPLACE PROCEDURE proc\_Grade (

p\_name IN VARCHAR2,

p\_total\_marks IN NUMBER

) AS

v\_class VARCHAR2(20);

BEGIN

-- Determine the category based on total marks

IF p\_total\_marks BETWEEN 990 AND 1500 THEN

v\_class := 'Distinction';

ELSIF p\_total\_marks BETWEEN 900 AND 989 THEN

v\_class := 'First Class';

ELSIF p\_total\_marks BETWEEN 825 AND 899 THEN

v\_class := 'Higher Second Class';

ELSE

v\_class := 'No Category';

END IF;

-- Insert the result into the Result table

INSERT INTO Result (Roll, name, Class)

VALUES ((SELECT NVL(MAX(Roll), 0) + 1 FROM Result), p\_name, v\_class);

DBMS\_OUTPUT.PUT\_LINE('Result added for student: ' || p\_name || ' with class: ' || v\_class);

EXCEPTION

WHEN OTHERS THEN

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

END proc\_Grade;

/

BEGIN

FOR student IN (SELECT name, total\_marks FROM Stud\_Marks) LOOP

-- Call proc\_Grade for each student

proc\_Grade(student.name, student.total\_marks);

END LOOP;

END;

/

P7.Create a stored function titled 'Age\_calc'. Accept the date of birth of a person as a parameter. Calculate the age of the person in years, months and days e.g. 3 years, 2months, 10 days. Return the age in years directly (with the help of Return statement). The months and days are to be returned indirectly in the form of OUT parameters.

CREATE OR REPLACE FUNCTION Age\_calc (

p\_dob DATE,

p\_months OUT NUMBER,

p\_days OUT NUMBER

) RETURN NUMBER AS

v\_years NUMBER;

v\_dob DATE := p\_dob;

v\_today DATE := SYSDATE;

BEGIN

-- Calculate the difference in years

v\_years := FLOOR(MONTHS\_BETWEEN(v\_today, v\_dob) / 12);

-- Calculate remaining months after full years

p\_months := FLOOR(MONTHS\_BETWEEN(v\_today, v\_dob) - (v\_years \* 12));

-- Calculate days remaining after full months

p\_days := v\_today - ADD\_MONTHS(v\_dob, v\_years \* 12 + p\_months);

RETURN v\_years; -- Return the age in years

END Age\_calc;

/

DECLARE

v\_years NUMBER;

v\_months NUMBER;

v\_days NUMBER;

BEGIN

-- Call the Age\_calc function with a sample date of birth

v\_years := Age\_calc(DATE '2000-05-15', v\_months, v\_days);

-- Display the results

DBMS\_OUTPUT.PUT\_LINE('Age: ' || v\_years || ' years, ' || v\_months || ' months, ' || v\_days || ' days');

END;

/

P8. Write a Row Level Before and After Trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library\_Audit table.

CREATE TABLE Library (

book\_id NUMBER PRIMARY KEY,

book\_name VARCHAR2(100),

author VARCHAR2(100),

published\_year NUMBER

);

CREATE TABLE Library\_Audit (

audit\_id NUMBER PRIMARY KEY,

action\_type VARCHAR2(10),

book\_id NUMBER,

book\_name VARCHAR2(100),

author VARCHAR2(100),

published\_year NUMBER,

action\_date DATE

);

CREATE SEQUENCE audit\_seq START WITH 1 INCREMENT BY 1;

CREATE OR REPLACE TRIGGER before\_update\_delete\_Library

BEFORE UPDATE OR DELETE ON Library

FOR EACH ROW

BEGIN

INSERT INTO Library\_Audit (

audit\_id,

action\_type,

book\_id,

book\_name,

author,

published\_year,

action\_date

)

VALUES (

audit\_seq.NEXTVAL,

CASE WHEN DELETING THEN 'DELETE' ELSE 'UPDATE' END,

:OLD.book\_id,

:OLD.book\_name,

:OLD.author,

:OLD.published\_year,

SYSDATE

);

END;

/

P9. Trigger: Create a row level trigger for the CUSTOMERS table that would fire INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values.

CREATE TABLE CUSTOMERS (

customer\_id NUMBER PRIMARY KEY,

customer\_name VARCHAR2(100),

salary NUMBER

);

CREATE OR REPLACE TRIGGER trg\_salary\_diff

BEFORE INSERT OR UPDATE OR DELETE ON CUSTOMERS

FOR EACH ROW

BEGIN

-- Display the salary for an INSERT operation

IF INSERTING THEN

DBMS\_OUTPUT.PUT\_LINE('New salary (INSERT): ' || :NEW.salary);

-- Display the salary difference for an UPDATE operation

ELSIF UPDATING THEN

IF :NEW.salary IS NOT NULL AND :OLD.salary IS NOT NULL THEN

DBMS\_OUTPUT.PUT\_LINE('Old salary: ' || :OLD.salary);

DBMS\_OUTPUT.PUT\_LINE('New salary: ' || :NEW.salary);

DBMS\_OUTPUT.PUT\_LINE('Salary difference: ' || (:NEW.salary - :OLD.salary));

ELSE

DBMS\_OUTPUT.PUT\_LINE('Salary update, but no previous salary available.');

END IF;

-- Display the salary for a DELETE operation

ELSIF DELETING THEN

DBMS\_OUTPUT.PUT\_LINE('Deleted salary: ' || :OLD.salary);

END IF;

END;

/

SET SERVEROUTPUT ON;

-- Insert example

INSERT INTO CUSTOMERS (customer\_id, customer\_name, salary) VALUES (1, 'John Doe', 5000);

-- Update example

UPDATE CUSTOMERS SET salary = 6000 WHERE customer\_id = 1;

-- Delete example

DELETE FROM CUSTOMERS WHERE customer\_id = 1;

P10.Trigger: Write a after trigger for Insert, update and delete event considering following requirement: Emp(Emp\_no, Emp\_name, Emp\_salary) a) Trigger should be initiated when salary tried to be inserted is less than Rs.50,000/- b) Trigger should be initiated when salary tried to be updated for value less than Rs. 50,000/- Also the new values expected to be inserted will be stored in new table Tracking(Emp\_no,Emp\_salary).

CREATE TABLE Emp (

Emp\_no NUMBER PRIMARY KEY,

Emp\_name VARCHAR2(100),

Emp\_salary NUMBER

);

CREATE TABLE Tracking (

Emp\_no NUMBER,

Emp\_salary NUMBER

);

CREATE OR REPLACE TRIGGER trg\_track\_low\_salary

AFTER INSERT OR UPDATE OR DELETE ON Emp

FOR EACH ROW

BEGIN

-- Insert condition: When inserting a new record with salary < 50,000

IF INSERTING THEN

IF :NEW.Emp\_salary < 50000 THEN

INSERT INTO Tracking (Emp\_no, Emp\_salary)

VALUES (:NEW.Emp\_no, :NEW.Emp\_salary);

DBMS\_OUTPUT.PUT\_LINE('Trigger fired: Insert with salary < 50,000. Record added to Tracking.');

END IF;

-- Update condition: When updating an existing record to a salary < 50,000

ELSIF UPDATING THEN

IF :NEW.Emp\_salary < 50000 THEN

INSERT INTO Tracking (Emp\_no, Emp\_salary)

VALUES (:NEW.Emp\_no, :NEW.Emp\_salary);

DBMS\_OUTPUT.PUT\_LINE('Trigger fired: Update with salary < 50,000. Record added to Tracking.');

END IF;

-- Delete condition: Track if you need to manage deleted data (optional, based on requirements)

ELSIF DELETING THEN

DBMS\_OUTPUT.PUT\_LINE('Delete operation detected, but no action required in this trigger.');

END IF;

END;

/

SET SERVEROUTPUT ON;

-- Test Insert

INSERT INTO Emp (Emp\_no, Emp\_name, Emp\_salary) VALUES (1, 'Alice', 45000);

-- Test Update

UPDATE Emp SET Emp\_salary = 40000 WHERE Emp\_no = 1;

-- Test Delete (no action in Tracking table for delete)

DELETE FROM Emp WHERE Emp\_no = 1;