**6 Create tables CitiesIndia(pincode,nameofcity,earliername,area,population,avgrainfall)**

**Categories(Type,pincode) *Note:- Enter data only in CitiesIndia***

**Write PL/SQL Procedure & function to find the population density of the cities. If the population density is above 3000 then Type of city must be entered as High Density in Category table. Between 2999 to 1000 as Moderate and below 999 as Low Density. Error must be displayed for population less than 10 or greater than 25718.**

-- Create database

CREATE DATABASE CityDB;

USE CityDB;

-- Create CitiesIndia table

CREATE TABLE CitiesIndia (

pincode INT PRIMARY KEY,

nameofcity VARCHAR(100) NOT NULL,

earliername VARCHAR(100),

area FLOAT NOT NULL,

population INT NOT NULL,

avgrainfall FLOAT

);

-- Create Categories table

CREATE TABLE Categories (

Type VARCHAR(20),

pincode INT PRIMARY KEY,

FOREIGN KEY (pincode) REFERENCES CitiesIndia(pincode)

);

-- Insert sample data into CitiesIndia

INSERT INTO CitiesIndia VALUES

(110001, 'Delhi', 'Dilli', 1484, 19000, 790),

(400001, 'Mumbai', 'Bombay', 603, 12442, 2420),

(560001, 'Bengaluru', 'Bangalore', 709, 8443, 970),

(700001, 'Kolkata', 'Calcutta', 185, 4486, 1600),

(600001, 'Chennai', NULL, 426, 4681, 1400);

-- Function to calculate population density

DELIMITER //

CREATE FUNCTION population\_density(pincode INT) RETURNS FLOAT

DETERMINISTIC

BEGIN

DECLARE dens FLOAT;

DECLARE pop INT;

DECLARE ar FLOAT;

SELECT population, area INTO pop, ar FROM CitiesIndia WHERE pincode = pincode;

IF pop < 10 OR pop > 25718 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Population out of valid range (10-25718)';

END IF;

SET dens = pop / ar;

RETURN dens;

END //

DELIMITER ;

-- Procedure to update Categories table based on population density

DELIMITER //

CREATE PROCEDURE UpdateCityCategory()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE c\_pincode INT;

DECLARE dens FLOAT;

DECLARE cur CURSOR FOR SELECT pincode FROM CitiesIndia;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO c\_pincode;

IF done THEN

LEAVE read\_loop;

END IF;

SET dens = population\_density(c\_pincode);

DELETE FROM Categories WHERE pincode = c\_pincode;

IF dens > 3000 THEN

INSERT INTO Categories VALUES ('High Density', c\_pincode);

ELSEIF dens BETWEEN 1000 AND 2999 THEN

INSERT INTO Categories VALUES ('Moderate Density', c\_pincode);

ELSE

INSERT INTO Categories VALUES ('Low Density', c\_pincode);

END IF;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

-- Run the procedure

CALL UpdateCityCategory();

-- Check data

SELECT \* FROM CitiesIndia;

SELECT \* FROM Categories;

**7 Write PL/SQL Procedure & function to find class [Distinction (Total marks from 1499 to 990) ,First Class( 899 to 900) Higher Second (899 to 825) ,Second,Pass (824 to 750) ] of a student based on total marks from table Student (rollno, name, Marks1, Marks2, Marks3, Marks4, Marks5).**

**Use exception handling when negative marks are entered by user(Marks<0) or Marks more than 100 are entered by user.. Store the result into Result table recording RollNo,total marks, and class for each student .**

-- Create database and use it

CREATE DATABASE IF NOT EXISTS StudentDB;

USE StudentDB;

-- Create Student table

CREATE TABLE Student (

rollno INT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

Marks1 INT,

Marks2 INT,

Marks3 INT,

Marks4 INT,

Marks5 INT

);

-- Create Result table

CREATE TABLE Result (

rollno INT PRIMARY KEY,

total\_marks INT,

class VARCHAR(20),

FOREIGN KEY (rollno) REFERENCES Student(rollno)

);

-- Insert sample data into Student table

INSERT INTO Student VALUES

(1, 'Rahul Sharma', 95, 98, 92, 88, 94),

(2, 'Neha Gupta', 78, 85, 80, 75, 82),

(3, 'Amit Singh', 65, 70, 60, 55, 68),

(4, 'Pooja Patel', 102, 90, 85, 88, 91), -- invalid marks (102)

(5, 'Karan Verma', -5, 88, 90, 84, 87); -- invalid marks (-5)

-- Function to calculate total marks and check for invalid marks

DELIMITER //

CREATE FUNCTION calculate\_total(roll INT) RETURNS INT

DETERMINISTIC

BEGIN

DECLARE total INT;

DECLARE m1, m2, m3, m4, m5 INT;

SELECT Marks1, Marks2, Marks3, Marks4, Marks5

INTO m1, m2, m3, m4, m5

FROM Student WHERE rollno = roll;

-- Check for invalid marks

IF m1 < 0 OR m1 > 100 OR m2 < 0 OR m2 > 100 OR m3 < 0 OR m3 > 100

OR m4 < 0 OR m4 > 100 OR m5 < 0 OR m5 > 100 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Invalid marks found (negative or greater than 100)';

END IF;

SET total = m1 + m2 + m3 + m4 + m5;

RETURN total;

END //

DELIMITER ;

-- Procedure to calculate class and insert into Result table

DELIMITER //

CREATE PROCEDURE Calculate\_Class()

BEGIN

DECLARE done INT DEFAULT 0;

DECLARE c\_roll INT;

DECLARE total\_marks INT;

DECLARE class\_result VARCHAR(20);

DECLARE cur CURSOR FOR SELECT rollno FROM Student;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END; -- skip errors (invalid marks)

OPEN cur;

read\_loop: LOOP

FETCH cur INTO c\_roll;

IF done THEN

LEAVE read\_loop;

END IF;

BEGIN

SET total\_marks = calculate\_total(c\_roll);

IF total\_marks BETWEEN 990 AND 1499 THEN

SET class\_result = 'Distinction';

ELSEIF total\_marks BETWEEN 900 AND 989 THEN

SET class\_result = 'First Class';

ELSEIF total\_marks BETWEEN 825 AND 899 THEN

SET class\_result = 'Higher Second';

ELSEIF total\_marks BETWEEN 750 AND 824 THEN

SET class\_result = 'Second';

ELSE

SET class\_result = 'Pass';

END IF;

-- Insert or update Result table

INSERT INTO Result (rollno, total\_marks, class) VALUES (c\_roll, total\_marks, class\_result)

ON DUPLICATE KEY UPDATE total\_marks = total\_marks, class = class\_result;

EXCEPTION

WHEN SQLEXCEPTION THEN

-- Skip this student if invalid marks

ITERATE read\_loop;

END;

END LOOP;

CLOSE cur;

END //

DELIMITER ;

-- Call the procedure to process students

CALL Calculate\_Class();

-- View results

SELECT \* FROM Result;

**29 Writ a PL/SQL procedure to find the number of students ranging from 100-70%, 69-60%, 59-50% & below 49% in each course from the student\_course table given by the procedure as parameter.**

**Schema: Student (ROLL\_NO ,COURSE, COURSE\_COD ,SEM ,TOTAL\_MARKS, PERCENTAGE)**

-- Create Student table

CREATE TABLE Student (

ROLL\_NO NUMBER PRIMARY KEY,

COURSE VARCHAR2(50),

COURSE\_COD VARCHAR2(10),

SEM NUMBER,

TOTAL\_MARKS NUMBER,

PERCENTAGE NUMBER(5,2)

);

-- Insert sample data (optional, for testing)

INSERT INTO Student VALUES (1, 'Math', 'MTH101', 1, 450, 75.0);

INSERT INTO Student VALUES (2, 'Math', 'MTH101', 1, 420, 70.0);

INSERT INTO Student VALUES (3, 'Math', 'MTH101', 1, 390, 65.0);

INSERT INTO Student VALUES (4, 'Physics', 'PHY101', 1, 350, 55.0);

INSERT INTO Student VALUES (5, 'Physics', 'PHY101', 1, 300, 48.0);

INSERT INTO Student VALUES (6, 'Chemistry', 'CHM101', 1, 280, 62.0);

INSERT INTO Student VALUES (7, 'Chemistry', 'CHM101', 1, 200, 40.0);

COMMIT;

-- Procedure to count students by percentage ranges for given course code

CREATE OR REPLACE PROCEDURE Count\_Students\_By\_Percentage (p\_course\_code IN VARCHAR2) IS

v\_100\_70 NUMBER;

v\_69\_60 NUMBER;

v\_59\_50 NUMBER;

v\_below\_49 NUMBER;

BEGIN

-- Count students with percentage between 70 and 100 (inclusive)

SELECT COUNT(\*) INTO v\_100\_70

FROM Student

WHERE COURSE\_COD = p\_course\_code AND PERCENTAGE BETWEEN 70 AND 100;

-- Count students with percentage between 60 and 69

SELECT COUNT(\*) INTO v\_69\_60

FROM Student

WHERE COURSE\_COD = p\_course\_code AND PERCENTAGE BETWEEN 60 AND 69;

-- Count students with percentage between 50 and 59

SELECT COUNT(\*) INTO v\_59\_50

FROM Student

WHERE COURSE\_COD = p\_course\_code AND PERCENTAGE BETWEEN 50 AND 59;

-- Count students with percentage below 50

SELECT COUNT(\*) INTO v\_below\_49

FROM Student

WHERE COURSE\_COD = p\_course\_code AND PERCENTAGE < 50;

-- Display results

DBMS\_OUTPUT.PUT\_LINE('Course Code: ' || p\_course\_code);

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

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

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

DBMS\_OUTPUT.PUT\_LINE('Students below 50%: ' || v\_below\_49);

END;

/

-- To enable output in SQL\*Plus or SQL Developer

-- SET SERVEROUTPUT ON;

-- Example call to the procedure

BEGIN

Count\_Students\_By\_Percentage('MTH101');

END;

/

**30 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 and900 category is first class, if marks 899 and 825 category is Higher Second Class .**

**Consider Schema as Stud\_Marks(name, total\_marks) and Result(Roll,Name, Class)**

-- Create tables Stud\_Marks and Result

CREATE TABLE Stud\_Marks (

Roll NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Total\_Marks NUMBER

);

CREATE TABLE Result (

Roll NUMBER PRIMARY KEY,

Name VARCHAR2(50),

Class VARCHAR2(30)

);

-- Insert sample data into Stud\_Marks (optional)

INSERT INTO Stud\_Marks VALUES (1, 'Amit', 1450);

INSERT INTO Stud\_Marks VALUES (2, 'Ravi', 920);

INSERT INTO Stud\_Marks VALUES (3, 'Sneha', 850);

INSERT INTO Stud\_Marks VALUES (4, 'Kiran', 700);

COMMIT;

-- Stored Procedure proc\_Grade to categorize students based on marks

CREATE OR REPLACE PROCEDURE proc\_Grade IS

BEGIN

-- Delete previous results to avoid duplicates

DELETE FROM Result;

-- Insert into Result based on category conditions

INSERT INTO Result (Roll, Name, Class)

SELECT Roll, Name,

CASE

WHEN Total\_Marks BETWEEN 990 AND 1500 THEN 'Distinction'

WHEN Total\_Marks BETWEEN 900 AND 989 THEN 'First Class'

WHEN Total\_Marks BETWEEN 825 AND 899 THEN 'Higher Second Class'

ELSE 'No Category'

END AS Class

FROM Stud\_Marks;

COMMIT;

END;

/

-- To run the procedure and see output

BEGIN

proc\_Grade;

END;

/

-- Check the Result table to verify categorization

SELECT \* FROM Result;

**9.PL/SQL code block: Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-**

**Schema:**

**1. Borrower(Rollin, Name, DateofIssue, NameofBook, Status)**

**2. Fine(Roll\_no,Date,Amt)**

**3. Library (bid, bname, doi, status,noc)**

**4. transaction (tid,bid, bname, status)**

1. **Accept roll\_no & name of book from user.**
2. **Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.**
3. **If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.**
4. **After submitting the book, status will change from I to R.**
5. **Update the noc in library according to the transaction made. Increase the noc if status is RETURN, Decrease noc if status is ISSUE.**
6. **If condition of fine is true, then details will be stored into fine table.**

-- 1. Create Tables

-- Borrower table to store student borrowing records

CREATE TABLE Borrower (

Rollin NUMBER PRIMARY KEY,

Name VARCHAR2(50),

DateofIssue DATE,

NameofBook VARCHAR2(100),

Status CHAR(1) -- 'I' for Issued, 'R' for Returned

);

-- Fine table to store fine details

CREATE TABLE Fine (

Roll\_no NUMBER,

Date DATE,

Amt NUMBER

);

-- Library table to store book copies and status

CREATE TABLE Library (

bid NUMBER PRIMARY KEY,

bname VARCHAR2(100),

doi DATE,

status CHAR(1), -- 'I' or 'R'

noc NUMBER -- Number of copies

);

-- Transaction table to record book issue/return activities

CREATE TABLE transaction (

tid NUMBER PRIMARY KEY,

bid NUMBER,

bname VARCHAR2(100),

status VARCHAR2(10) -- 'ISSUE' or 'RETURN'

);

-- 2. Insert Dummy Data

INSERT INTO Borrower VALUES (101, 'Rahul', TO\_DATE('2024-04-15','YYYY-MM-DD'), 'DBMS', 'I');

INSERT INTO Borrower VALUES (102, 'Priya', TO\_DATE('2024-04-01','YYYY-MM-DD'), 'OS', 'I');

INSERT INTO Library VALUES (1, 'DBMS', TO\_DATE('2020-01-01','YYYY-MM-DD'), 'I', 2);

INSERT INTO Library VALUES (2, 'OS', TO\_DATE('2020-01-01','YYYY-MM-DD'), 'I', 3);

INSERT INTO transaction VALUES (1001, 1, 'DBMS', 'ISSUE');

INSERT INTO transaction VALUES (1002, 2, 'OS', 'ISSUE');

-- 3. PL/SQL Block to Handle Return, Fine Calculation, and Updating Library

SET SERVEROUTPUT ON;

DECLARE

v\_roll Borrower.Rollin%TYPE := &roll\_no; -- Input: Roll Number

v\_book Borrower.NameofBook%TYPE := '&book\_name'; -- Input: Book Name

v\_date DATE;

v\_days NUMBER;

v\_fine NUMBER := 0;

v\_bid NUMBER;

v\_noc NUMBER;

BEGIN

-- Get date of issue for this student and book

SELECT DateofIssue INTO v\_date

FROM Borrower

WHERE Rollin = v\_roll AND NameofBook = v\_book;

-- Calculate number of days since issue

v\_days := SYSDATE - v\_date;

-- Fine logic

IF v\_days BETWEEN 15 AND 30 THEN

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

ELSIF v\_days > 30 THEN

v\_fine := v\_days \* 50;

END IF;

-- Update Borrower table status to Returned

UPDATE Borrower

SET Status = 'R'

WHERE Rollin = v\_roll AND NameofBook = v\_book;

-- Get book ID and current number of copies from Library

SELECT bid, noc INTO v\_bid, v\_noc

FROM Library

WHERE bname = v\_book;

-- Increase number of copies in Library

UPDATE Library

SET noc = v\_noc + 1

WHERE bid = v\_bid;

-- Insert fine if applicable

IF v\_fine > 0 THEN

INSERT INTO Fine VALUES (v\_roll, SYSDATE, v\_fine);

END IF;

-- Show output

DBMS\_OUTPUT.PUT\_LINE('Book returned successfully.');

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

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No record found for given Roll No and Book.');

WHEN OTHERS THEN

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

END;

/

**26 Write a PL/SQL code to calculate tax for an employee of an organization ABC and to display his/her name & tax, by creating a table under employee database as below:**

**Employee\_salary(emp\_no,basic,HRA,DA,Total\_deduction,net\_salary,gross\_Salary)**

-- 1. Create Employee\_salary table

CREATE TABLE Employee\_salary (

emp\_no NUMBER PRIMARY KEY,

basic NUMBER NOT NULL,

HRA NUMBER NOT NULL,

DA NUMBER NOT NULL,

Total\_deduction NUMBER NOT NULL,

net\_salary NUMBER,

gross\_salary NUMBER

);

-- 2. Insert sample data

INSERT INTO Employee\_salary VALUES (101, 50000, 10000, 5000, 2000, NULL, NULL);

INSERT INTO Employee\_salary VALUES (102, 30000, 7000, 3000, 1500, NULL, NULL);

INSERT INTO Employee\_salary VALUES (103, 40000, 8000, 4000, 1800, NULL, NULL);

-- Commit data

COMMIT;

-- 3. PL/SQL block to calculate gross salary, net salary, tax and display results

DECLARE

v\_emp\_no Employee\_salary.emp\_no%TYPE;

v\_basic Employee\_salary.basic%TYPE;

v\_HRA Employee\_salary.HRA%TYPE;

v\_DA Employee\_salary.DA%TYPE;

v\_Total\_deduction Employee\_salary.Total\_deduction%TYPE;

v\_gross\_salary NUMBER;

v\_net\_salary NUMBER;

v\_tax NUMBER;

BEGIN

-- Cursor to fetch employee details

FOR emp\_rec IN (SELECT emp\_no, basic, HRA, DA, Total\_deduction FROM Employee\_salary) LOOP

v\_emp\_no := emp\_rec.emp\_no;

v\_basic := emp\_rec.basic;

v\_HRA := emp\_rec.HRA;

v\_DA := emp\_rec.DA;

v\_Total\_deduction := emp\_rec.Total\_deduction;

-- Calculate gross salary

v\_gross\_salary := v\_basic + v\_HRA + v\_DA;

-- Calculate net salary

v\_net\_salary := v\_gross\_salary - v\_Total\_deduction;

-- Calculate tax based on net salary slabs (example):

IF v\_net\_salary <= 25000 THEN

v\_tax := v\_net\_salary \* 0.05; -- 5% tax

ELSIF v\_net\_salary <= 50000 THEN

v\_tax := v\_net\_salary \* 0.1; -- 10% tax

ELSE

v\_tax := v\_net\_salary \* 0.15; -- 15% tax

END IF;

-- Update gross\_salary and net\_salary in the table

UPDATE Employee\_salary

SET gross\_salary = v\_gross\_salary,

net\_salary = v\_net\_salary

WHERE emp\_no = v\_emp\_no;

-- Display employee no and calculated tax

DBMS\_OUTPUT.PUT\_LINE('Employee No: ' || v\_emp\_no || ' - Tax: ' || TO\_CHAR(v\_tax, 'FM999999.00'));

END LOOP;

-- Commit the updates

COMMIT;

EXCEPTION

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

**27 Create PL/SQL code block: Write a PL/SQL block of code for the following schema:**

**Borrower(Rollin, Name, DateofIssue, NameofBook, Status)**

**Fine(Roll\_no,Date,Amt)**

**Solve following queries:**

1. **Accept roll\_no & name of book from user.**
2. **Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.**
3. **If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.**
4. **After submitting the book, status will change from I to R.**
5. **If condition of fine is true, then details will be stored into fine table.**

**Use of Control structure and Exception handling is mandatory.**

-- Create Borrower table

CREATE TABLE Borrower (

Rollin NUMBER PRIMARY KEY,

Name VARCHAR2(50),

DateofIssue DATE,

NameofBook VARCHAR2(100),

Status CHAR(1) -- 'I' = Issued, 'R' = Returned

);

-- Create Fine table

CREATE TABLE Fine (

Roll\_no NUMBER,

Date DATE,

Amt NUMBER

);

-- Insert sample data

INSERT INTO Borrower VALUES (1, 'Ravi', TO\_DATE('2024-04-01', 'YYYY-MM-DD'), 'C Programming', 'I');

INSERT INTO Borrower VALUES (2, 'Sneha', TO\_DATE('2024-04-20', 'YYYY-MM-DD'), 'Java', 'I');

-- Enable output

SET SERVEROUTPUT ON;

-- Simplified PL/SQL Block

DECLARE

v\_rollno NUMBER := &Enter\_Roll\_Number; -- User will input Roll Number

v\_book VARCHAR2(100) := '&Enter\_Book\_Name'; -- User will input Book Name

v\_issue\_date DATE;

v\_days NUMBER;

v\_fine NUMBER := 0;

BEGIN

-- Get issue date from Borrower table

SELECT DateofIssue INTO v\_issue\_date

FROM Borrower

WHERE Rollin = v\_rollno AND NameofBook = v\_book AND Status = 'I';

-- Calculate number of days since book was issued

v\_days := SYSDATE - v\_issue\_date;

-- Fine calculation

IF v\_days > 30 THEN

v\_fine := v\_days \* 50;

ELSIF v\_days >= 15 THEN

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

END IF;

-- Update book status to Returned

UPDATE Borrower

SET Status = 'R'

WHERE Rollin = v\_rollno AND NameofBook = v\_book;

-- Insert fine if applicable

IF v\_fine > 0 THEN

INSERT INTO Fine VALUES (v\_rollno, SYSDATE, v\_fine);

END IF;

-- Show result

DBMS\_OUTPUT.PUT\_LINE('Book returned.');

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

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

-- Handle errors

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('No such issued book found.');

WHEN OTHERS THEN

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

END;

/

**28 Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N\_RollCall 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.**

-- Step 1: Create the original roll call table

CREATE TABLE O\_RollCall (

Roll\_No NUMBER PRIMARY KEY,

Name VARCHAR2(50)

);

-- Step 2: Create the new roll call table

CREATE TABLE N\_RollCall (

Roll\_No NUMBER,

Name VARCHAR2(50)

);

-- Step 3: Insert sample data into O\_RollCall (old data)

INSERT INTO O\_RollCall VALUES (1, 'Ravi');

INSERT INTO O\_RollCall VALUES (2, 'Sneha');

-- Step 4: Insert sample data into N\_RollCall (new data to merge)

INSERT INTO N\_RollCall VALUES (2, 'Sneha'); -- Duplicate

INSERT INTO N\_RollCall VALUES (3, 'Ankit'); -- New

INSERT INTO N\_RollCall VALUES (4, 'Meena'); -- New

-- Step 5: PL/SQL block to merge data using parameterized cursor

SET SERVEROUTPUT ON;

DECLARE

-- Cursor to fetch each row from N\_RollCall based on Roll\_No

CURSOR new\_data\_cursor(p\_roll N\_RollCall.Roll\_No%TYPE) IS

SELECT Roll\_No, Name FROM N\_RollCall WHERE Roll\_No = p\_roll;

-- Variables to hold cursor values

v\_roll N\_RollCall.Roll\_No%TYPE;

v\_name N\_RollCall.Name%TYPE;

BEGIN

-- Loop through all new records

FOR rec IN (SELECT \* FROM N\_RollCall) LOOP

BEGIN

-- Check if Roll\_No already exists in O\_RollCall

SELECT Roll\_No INTO v\_roll

FROM O\_RollCall

WHERE Roll\_No = rec.Roll\_No;

-- If found, skip insertion

DBMS\_OUTPUT.PUT\_LINE('Skipping duplicate Roll\_No: ' || rec.Roll\_No);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

-- If not found, insert the new record into O\_RollCall

INSERT INTO O\_RollCall (Roll\_No, Name)

VALUES (rec.Roll\_No, rec.Name);

DBMS\_OUTPUT.PUT\_LINE('Inserted: Roll\_No = ' || rec.Roll\_No || ', Name = ' || rec.Name);

END;

END LOOP;

END;

/