**Experiment-7**

**Aim:**

Program development using WHILE LOOPS, numeric FOR LOOPS, nested loops using ERROR Handling, BUILT –IN Exceptions, USE defined Exceptions, RAISE- APPLICATION ERROR.

**Description**

**WHILE Loops**

A WHILE loop is used to execute a block of code **repeatedly** as long as a specified condition remains TRUE. It is useful when the number of iterations is not known beforehand.

**Syntax:**

WHILE condition LOOP

-- Code to be executed

END LOOP;

* The loop checks the condition **before** executing the block.
* If the condition is FALSE at the start, the loop **never runs**.

**Numeric FOR Loops**

A FOR loop is used when the number of iterations is **known in advance**. It executes a block **a fixed number of times** based on a specified range.

**Syntax:**

FOR counter\_variable IN start\_value..end\_value LOOP

-- Code to be executed

END LOOP;

* The loop automatically increments the counter from start\_value to end\_value.
* The counter variable is implicitly declared and cannot be modified inside the loop.

**Nested Loops**

A **nested loop** means placing one loop inside another. Each time the outer loop runs once, the inner loop executes completely.

**Syntax:**

FOR i IN 1..3 LOOP

FOR j IN 1..5 LOOP

-- Code to be executed

END LOOP;

END LOOP;

* Used for **processing multi-dimensional data**, such as generating tables or working with matrices.
* The **inner loop executes fully** for each iteration of the **outer loop**.

**Error Handling in PL/SQL**

Error handling ensures that a program does not fail abruptly but instead handles exceptions gracefully. There are two main types:

**A. Built-in Exceptions**

PL/SQL provides predefined exceptions for common errors, such as:

* NO\_DATA\_FOUND – Raised when a query returns no rows.
* ZERO\_DIVIDE – Raised when an attempt is made to divide by zero.
* TOO\_MANY\_ROWS – Raised when a SELECT INTO statement returns multiple rows instead of one.

**B. User-Defined Exceptions**

PL/SQL allows developers to define their own exceptions when built-in exceptions are not sufficient.

**Syntax:**

DECLARE

custom\_exception EXCEPTION;

BEGIN

IF some\_condition THEN

RAISE custom\_exception;

END IF;

EXCEPTION

WHEN custom\_exception THEN

DBMS\_OUTPUT.PUT\_LINE('Custom error occurred.');

END;

* The exception is **declared** in the DECLARE section.
* It is **raised** using RAISE when a specific condition is met.
* It is **handled** in the EXCEPTION block.

**RAISE\_APPLICATION\_ERROR**

RAISE\_APPLICATION\_ERROR is used to **generate a custom error message** with a specific error code in the range of -20000 to -20999.

**Syntax:**

RAISE\_APPLICATION\_ERROR(-20001, 'A custom error message.');

* Used when a critical issue requires immediate program termination.
* Helps in debugging and providing meaningful error messages.

**Program**

CREATE TABLE studenttts (

student\_id NUMBER PRIMARY KEY,

student\_name VARCHAR2(100),

score NUMBER);

INSERT INTO studenttts VALUES (1, 'Alice', 85);

INSERT INTO studenttts VALUES (2, 'Bob', 92);

INSERT INTO studenttts VALUES (3, 'Charlie', NULL);

INSERT INTO studenttts VALUES (4, 'David', -5);

INSERT INTO studenttts VALUES (5, 'Eve', 70);

COMMIT;

SET SERVEROUTPUT ON;

DECLARE

TYPE Scores IS TABLE OF NUMBER;

student\_scores Scores := Scores();

total\_score NUMBER := 0;

average\_score NUMBER;

i NUMBER := 1;

invalid\_score EXCEPTION;

division\_by\_zero EXCEPTION;

total\_students NUMBER;

BEGIN

-- Fetch data into the collection

SELECT score BULK COLLECT INTO student\_scores FROM studentts;

IF student\_scores.COUNT = 0 THEN

RAISE NO\_DATA\_FOUND;

END IF;

total\_students := student\_scores.COUNT;

-- WHILE loop to calculate the total score, skipping invalid scores instead of stopping execution

WHILE i <= total\_students LOOP

IF student\_scores(i) IS NULL THEN

DBMS\_OUTPUT.PUT\_LINE('Score is NULL for student ' || i || '. Skipping...');

ELSIF student\_scores(i) < 0 THEN

--- DBMS\_OUTPUT.PUT\_LINE('Invalid score encountered for student ' || i || '. Skipping...');

RAISE invalid\_score;

ELSE

total\_score := total\_score + student\_scores(i);

END IF;

i := i + 1;

END LOOP;

-- Calculate and display total and average score

IF total\_students = 0 OR total\_score = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('No valid scores found. Skipping average calculation.');

ELSE

average\_score := total\_score / total\_students;

DBMS\_OUTPUT.PUT\_LINE('Total Score: ' || total\_score);

DBMS\_OUTPUT.PUT\_LINE('Average Score: ' || average\_score);

END IF;

-- Display multiplication table

DBMS\_OUTPUT.PUT\_LINE('Multiplication Table:');

FOR j IN 1..3 LOOP

FOR k IN 1..5 LOOP

DBMS\_OUTPUT.PUT\_LINE(j || ' x ' || k || ' = ' || (j \* k));

END LOOP;

END LOOP;

EXCEPTION

WHEN invalid\_score THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Invalid score encountered. All scores should be non-negative.');

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: No data found.');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20001, 'An unknown error occurred.');

END;

/