Ex.No: 8 ***Cursor management, Creation of Triggeres and Exceptions in SQL***

Aim:

To practice and implement the concepts of cursor management in different ways so as to get a complete knowledge about the implementation of cursors

Description:

Cursor:

A cursor is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the active set.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors –

- Implicit cursors
- Explicit cursors

Program to write a Cursor to list all the Administrators in the Database Table named as Admin

Output:

```
SQL> DECLARE
  2    e_id employee.emp_id%TYPE;
  3    e_name employee.emp_name%TYPE;
  4      CURSOR e_emp IS
  5      SELECT emp_id, emp_name FROM employee;
  6  BEGIN
  7    OPEN e_emp;
  8    LOOP
  9      FETCH e_emp INTO e_id, e_name; -- Fetch only the columns selected by the cursor
 10      EXIT WHEN e_emp%NOTFOUND;
 11          DBMS_OUTPUT.PUT_LINE(e_id || ' ' || e_name);   END LOOP;
 12    CLOSE e_emp;
 13  END;
 14  /
200 SUKUNA
201 MADARA
202 VEGETA
203 ICHIGO

PL/SQL procedure successfully completed.
```

Update the table and increase the salary of each customer by 500 and use the SQL%ROWCOUNT attribute to determine the number of rows affected

```
SQL>  DECLARE
  2    total_rows NUMBER(2);
  3  BEGIN
  4    UPDATE Emp
  5    SET salary = salary + 500;
  6
  7    IF SQL%NOTFOUND THEN
  8      DBMS_OUTPUT.PUT_LINE('No employee selected');
  9    ELSIF SQL%FOUND THEN
 10      total_rows := SQL%ROWCOUNT;
 11      DBMS_OUTPUT.PUT_LINE(total_rows || ' Employee(s) selected');
 12    END IF;
 13  END;
 14  /
3 Employee(s) selected

PL/SQL procedure successfully completed.

SQL>
SQL> select * from emp;

EMP_I ENAME      EMAIL_ID        PHONE     SALARY SHI
----- ---------- ---------- ---------- ---------- ---
123   Rithi      rith@gmail 9003402381      38500
456   Harini     hari@gmail 9345542103      58500 nyt
678   Raji       raju@gmail  723456890      78500 day
```

Explicit Cursors

The syntax for creating an explicit cursor is –

```
CURSOR cursor_name IS select_statement;
```

Declaring the Cursor

```
CURSOR c_customers IS
  SELECT id, name, address FROM customers;
```

Opening the Cursor

```
OPEN c_customers;
```

Fetching the Cursor

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

```
CLOSE c_customers;
```

Output:

```
SQL> DECLARE
  2   CURSOR vehicle_cursor IS
  3        SELECT colour, COUNT(*) AS num_vehicles
  4        FROM vehicle_tb
  5        GROUP BY colour;
  6     -- Variables to store cursor values
  7     v_colour vehicle_tb.colour%TYPE;
  8     v_num NUMBER;
  9  BEGIN
 10     OPEN vehicle_cursor;
 11     LOOP
 12        FETCH vehicle_cursor INTO v_colour, v_num;
 13        EXIT WHEN vehicle_cursor%NOTFOUND;
 14
 15        DBMS_OUTPUT.PUT_LINE('Color: ' || v_colour || ', Number
of Vehicles: ' || v_num);
 16     END LOOP;
 17     CLOSE vehicle_cursor;
 18  END;
 19
 20  /
Color: Blue, Number of Vehicles: 2
Color: Brown, Number of Vehicles: 1
Color: Black, Number of Vehicles: 1

PL/SQL procedure successfully completed.

SQL>
```

Triggers:

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events –

- **A database manipulation (DML**) statement (DELETE, INSERT, or UPDATE)
- **A database definition (DDL)** statement (CREATE, ALTER, or DROP).
- **A database operation** (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Creating Triggers

The syntax for creating a trigger is –

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n] [FOR
EACH ROW]
WHEN (condition)
DECLARE
  Declaration-statements BEGIN
  Executable-statements EXCEPTION
  Exception-handling-statements END;
```

Output:

```
SQL> CREATE TRIGGER pay_trig
  2  AFTER
  3  INSERT
  4  ON Payment
  5  FOR EACH ROW
  6  WHEN(NEW.Payment_ID>0)
  7  DECLARE
  8  BEGIN
  9  dbms_output.put_line('NEW RECORD CREATED');
 10  END;
 11  /

Trigger created.
```

```
SQL> insert INTO Payment values(56899,85459,5163,'19/3/24','CASH');
NEW RECORD CREATED

1 row created.
```

```
SQL> CREATE OR REPLACE TRIGGER rental_changes
  2  AFTER DELETE OR INSERT OR UPDATE ON VEHICLE_TB
  3  FOR EACH ROW
  4  WHEN (NEW.VEHICLE_ID > 0)
  5  DECLARE
  6    rental_diff NUMBER;
  7  BEGIN
  8    rental_diff := :NEW.RENTAL_RATE - :OLD.RENTAL_RATE;
  9    dbms_output.put_line('New rental rate: ' || :NEW.rental_rate);
 10    dbms_output.put_line('Old rental rate: ' || :OLD.rental_rate);
 11    dbms_output.put_line('Rental change: ' || rental_diff);
 12  END;
 13  /

Trigger created.

SQL> update vehicle_tb set rental_rate=5000 where vehicle_id=2;

1 row updated.

SQL> SET SERVEROUTPUT ON;
SQL> update vehicle_tb set rental_rate=5000 where vehicle_id=2;
New rental rate: 5000
Old rental rate: 5000
Rental change: 0
```

Exception:

- System-defined exceptions

Syntax for Exception Handling

WHEN others THEN –

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling goes here >
  WHEN exception1 THEN exception1-
    handling-statements.
  WHEN exception2  THEN exception2-
    handling-statements
  WHEN exception3 THEN exception3-
    handling-statements
  ........
  WHEN others THEN exception-
handling-statements; end;
```

Output:

```
SQL> DECLARE
  2    v_Model vehicles.MODEL%type;
  3    v_Manufacture vehicles.MANUFACTURE%type;
  4    v_Color vehicles.COLOR%type;
  5    v_Mileage vehicles.MILEAGE%type;
  6    v_RegNo vehicles.REG_NO%type := 56899; -- Example registration number
  7
  8  BEGIN
  9    SELECT MODEL, MANUFACTURE, COLOR, MILEAGE
 10    INTO v_Model, v_Manufacture, v_Color, v_Mileage
 11    FROM vehicles
 12    WHERE REG_NO = v_RegNo;
 13
 14    DBMS_OUTPUT.PUT_LINE('Vehicle Details:');
 15    DBMS_OUTPUT.PUT_LINE('Model: ' || v_Model);
 16    DBMS_OUTPUT.PUT_LINE('Manufacture: ' || v_Manufacture);
 17    DBMS_OUTPUT.PUT_LINE('Color: ' || v_Color);
 18    DBMS_OUTPUT.PUT_LINE('Mileage: ' || v_Mileage);
 19
 20  EXCEPTION
 21    WHEN NO_DATA_FOUND THEN
 22      DBMS_OUTPUT.PUT_LINE('No vehicle found with registration number: ' || v_RegNo);
 23    WHEN OTHERS THEN
 24      DBMS_OUTPUT.PUT_LINE('An error occurred while retrieving vehicle details.');
 25  END;
 26  /

PL/SQL procedure successfully completed.

SQL>
```

User Defined Exception

Raising Exceptions

Syntax for raising an exception –

```
DECLARE
  exception_name EXCEPTION;
BEGIN
  IF condition THEN
    RAISE exception_name;
  END IF;
EXCEPTION
  WHEN exception_name THEN
  statement;
END;
```

Output:

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2      -- Define custom exceptions
  3      vehicle_not_found EXCEPTION;
  4      mileage_above_threshold EXCEPTION;
  5
  6      -- Declare variables
  7      v_Model vehicles.MODEL%type;
  8      v_Manufacture vehicles.MANUFACTURE%type;
  9      v_Color vehicles.COLOR%type;
 10      v_Mileage vehicles.MILEAGE%type;
 11      v_RegNo vehicles.REG_NO%type := 56899; -- Example registration number
 12
 13  BEGIN
 14      -- Attempt to retrieve vehicle details
 15      SELECT MODEL, MANUFACTURE, COLOR, MILEAGE
 16      INTO v_Model, v_Manufacture, v_Color, v_Mileage
 17      FROM vehicles
 18      WHERE REG_NO = v_RegNo;
 19
 20      -- Output vehicle details if found
 21      DBMS_OUTPUT.PUT_LINE('Vehicle Details:');
 22      DBMS_OUTPUT.PUT_LINE('Model: ' || v_Model);
 23      DBMS_OUTPUT.PUT_LINE('Manufacture: ' || v_Manufacture);
 24      DBMS_OUTPUT.PUT_LINE('Color: ' || v_Color);
 25
 26      -- Check if mileage is NULL or above a certain threshold
 27      IF v_Mileage IS NULL THEN
 28          DBMS_OUTPUT.PUT_LINE('Mileage information not available.');
 29      ELSIF v_Mileage > 100000 THEN
 30          RAISE mileage_above_threshold;
 31      ELSE
 32          DBMS_OUTPUT.PUT_LINE('Mileage: ' || v_Mileage);
 33      END IF;
 34
 35  EXCEPTION
 36      WHEN NO_DATA_FOUND THEN
 37          -- Raise custom exception if vehicle is not found
 38          raise_application_error(-20001, 'No vehicle found with registration number: ' || v_RegNo);
 39      WHEN mileage_above_threshold THEN
 40          DBMS_OUTPUT.PUT_LINE('Mileage exceeds threshold.');
 41      WHEN OTHERS THEN
 42          DBMS_OUTPUT.PUT_LINE('An error occurred while retrieving vehicle details.');
 43  END;
 44  /
DECLARE
*
ERROR at line 1:
ORA-20001: No vehicle found with registration number: 56899
ORA-06512: at line 38
```

DBMS_STANDARD.RAISE_APPLICATION_ERROR.

Output:

```
SQL> DECLARE
  2      v_RegNo vehicles.REG_NO%type := 56899;
  3      v_model vehicles.MODEL%TYPE; -- Declaration of v_model variable
  4  BEGIN
  5      -- Attempt to retrieve vehicle details
  6      SELECT MODEL
  7      INTO v_Model
  8      FROM vehicles
  9      WHERE REG_NO = v_RegNo;
 10
 11      IF v_Model IS NULL THEN
 12          -- Incorrect usage of RAISE_APPLICATION_ERROR with a custom error code
 13          DBMS_STANDARD.RAISE_APPLICATION_ERROR(-20002, 'Vehicle details not found.');
 14      END IF;
 15
 16  EXCEPTION
 17      WHEN NO_DATA_FOUND THEN
 18          -- Correct usage of RAISE_APPLICATION_ERROR with a meaningful error message
 19          DBMS_STANDARD.RAISE_APPLICATION_ERROR(-20001, 'No vehicle found with registration number: ' || v_RegNo);
 20      WHEN OTHERS THEN
 21          -- Incorrect usage of RAISE_APPLICATION_ERROR without a specific error condition
 22          DBMS_STANDARD.RAISE_APPLICATION_ERROR(-20003, 'An unexpected error occurred.');
 23  END;
 24  /
DECLARE
*
ERROR at line 1:
ORA-20001: No vehicle found with registration number: 56899
ORA-06512: at line 19
```

Result

Thus the implementation of cursor, triggers and exceptions have been done and verified successfully