MIT World Peace University

Database Management System

Assignment 8

Naman Soni Roll No. 10

Contents

1 Aim **1**

2 Objectives **1**

3 Problem Statement **1**

4 Theory **1**

4.1 What is Cursor? 1
4.2 Why do we need the Cursors? 1
4.3 Different types of Cursors 2
4.4 Drawbacks of Implicit Cursors 2
4.5 PL/SQL variables in a Cursor 2
4.6 Opening a Cursor 2
4.7 Fetching from a Cursor 2
4.8 Closing a Cursor 3
4.9 Cursor attributes 3

5 Platform **3**

6 Input **3**

7 Queries **4**

8 Outputs in Execution **5**

9 Conclusion **8**

10 FAQ **9**

1 Aim

Write PL/SQL Cursor for the given problem statements

2 Objectives

1. To study and use Cursor using MySQL PL/SQL block

3 Problem Statement

Create tables and solve given queries.

4 Theory

Explain following points

4.1 What is Cursor?

A cursor is a database object that allows a program to retrieve data from a database one row at a time. It acts like a pointer that points to a specific row in a result set returned by a SQL query. Cursors are useful in situations where we need to manipulate data row by row, such as in a loop. They allow us to navigate through the rows of a result set and manipulate them as needed.

Example:

```
DECLARE
    CURSOR c1 IS
        SELECT * FROM employees;
BEGIN
    FOR employee IN c1 LOOP
        DBMS_OUTPUT.PUT_LINE(employee.first_name || ' ' || employee.last_name)
;
    END LOOP;
END;
```

4.2 Why do we need the Cursors?

Cursors are necessary in situations where we need to retrieve data from a database one row at a time and process it. For example, if we want to perform a complex calculation on a large dataset, it may be more efficient to retrieve the data row by row rather than all at once. Cursors enable us to retrieve data one row at a time and process it as needed, which can be particularly useful in situations where we need to perform complex processing on data.

1. Cursors are used to retrieve data from a database one row at a time.
2. Cursors are used to process data row by row.
3. Cursors are used to perform complex calculations on large datasets.

4.3 Different types of Cursors

There are two types of cursors in PL/SQL:

1. Implicit Cursor: This is a cursor that is automatically created by the Oracle database when a SQL statement is executed. It is used for simple, one-time queries that return a small number of rows.
2. Explicit Cursor: This is a cursor that is explicitly declared and defined in a PL/SQL program. It is used for more complex queries that require multiple rows to be returned and processed.

4.4 Drawbacks of Implicit Cursors

Implicit cursors have a few drawbacks, including:

1. They do not provide much control over the result set returned by the query.
2. They do not allow us to manipulate data row by row.
3. They can cause performance issues if used in a loop or if the query returns a large number of rows.

4.5 PL/SQL variables in a Cursor

PL/SQL variables can be used in a cursor to pass values into the query. For example, we can declare a variable in the cursor declaration and use it in the WHERE clause of the query. This can be particularly useful when we need to retrieve a subset of data from a larger dataset.

4.6 Opening a Cursor

Before we can use a cursor, we need to open it using the OPEN statement. This statement prepares the cursor for fetching data from the result set. Once the cursor is open, we can begin fetching data from it.

```
    DECLARE
        CURSOR c1 IS
            SELECT * FROM employees;
    BEGIN
        OPEN c1;
        ...
    END;
```

4.7 Fetching from a Cursor

To retrieve data from a cursor, we use the FETCH statement. This statement retrieves the next row from the result set and makes it available for processing. We can fetch data from the cursor one row at a time, or we can retrieve all of the rows at once.

```
    DECLARE
        CURSOR c1 IS
            SELECT * FROM employees;
        emp_row employees%ROWTYPE;
    BEGIN
        OPEN c1;
```

```
        FETCH c1 INTO emp_row;
        ...
    END;
```

4.8 Closing a Cursor

Once we are done processing the result set, we need to close the cursor using the CLOSE statement. This statement releases the resources used by the cursor and frees up memory. It is important to close cursors when we are done using them to avoid wasting system resources.

```
    DECLARE
        CURSOR c1 IS
            SELECT * FROM employees;
        emp_row employees%ROWTYPE;
    BEGIN
        OPEN c1;
        FETCH c1 INTO emp_row;
        CLOSE c1;
    END;
```

4.9 Cursor attributes

Cursor attributes are properties of a cursor that provide information about its status. Some examples of cursor attributes include %FOUND, which indicates whether the last fetch retrieved a row, and %NOTFOUND, which indicates whether the last fetch did not retrieve a row.

```
  DECLARE
    CURSOR employee_cursor IS
      SELECT * FROM employees;
    emp_row employees%ROWTYPE;
  BEGIN
    OPEN employee_cursor;
    LOOP
      FETCH employee_cursor INTO emp_row;
      EXIT WHEN employee_cursor%NOTFOUND;
      -- process data here
    END LOOP;
    CLOSE employee_cursor;
  END;
```

5 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Draw.io for Drawing the ER diagram.

6 Input

Given Database from the Problem Statement for the Assignment for our batch. (A1 PA 20)

7 Queries

```

-- Use a cursor to calculate compound interest for each customer
-- and insert customer id and simple interest in another table
-- named TEMPLIST.
-- Customer(cust_id, Prinicipal_amount, Rate_of_interest, No. of
-- Years)

-- Create Database and Tables

CREATE database if not exists store;
use store;
CREATE TABLE BOOK (
    Isbn VARCHAR(10) PRIMARY KEY,
    Title VARCHAR(100),
    SoldCopies INT
);

CREATE TABLE WRITING (
    Isbn VARCHAR(10),
    Name VARCHAR(50),
    PRIMARY KEY (Isbn, Name),
    FOREIGN KEY (Isbn) REFERENCES BOOK(Isbn)
);

CREATE TABLE AUTHOR (
    Name VARCHAR(50) PRIMARY KEY,
    SoldCopies INT
);

CREATE TABLE Customer (
    cust_id INT PRIMARY KEY,
    Principal_amount DOUBLE,
    Rate_of_interest DOUBLE,
    Years INT
);


INSERT INTO BOOK (Isbn, Title, SoldCopies)
VALUES ('9783161', 'Crime and Punishment', 500);

INSERT INTO WRITING (Isbn, Name)
VALUES ('9783161', 'Fyodor Dostoevsky');

INSERT INTO AUTHOR (Name, SoldCopies)
VALUES ('Fyodor Dostoevsky', 500);

INSERT INTO Customer (cust_id, Principal_amount, Rate_of_interest, Years)
VALUES (1, 1000, 0.05, 5);


-- Create Cursors
USE store;

DROP TABLE IF EXISTS TEMPLIST;
CREATE TABLE TEMPLIST (
  cust_id INT PRIMARY KEY,
```

```
  Compound_interest DOUBLE
);

DROP PROCEDURE IF EXISTS calculate_interest;
DELIMITER //
CREATE PROCEDURE calculate_interest()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE custid INT;
  DECLARE princ_amt, rate, num_years, interest_amt, temp_amt DOUBLE;
  DECLARE cur CURSOR FOR SELECT cust_id, Principal_amount, Rate_of_interest, Years
     FROM Customer;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur;
  loop1: LOOP
    FETCH cur INTO custid, princ_amt, rate, num_years;
    IF done THEN
      LEAVE loop1;
    END IF;

    SET temp_amt = 1 + rate / 12;
    SET interest_amt = princ_amt * POWER(temp_amt, num_years * 12) - princ_amt;

    INSERT INTO TEMPLIST (cust_id, Compound_interest) VALUES (custid, interest_amt
    );
  END LOOP;
  CLOSE cur;
END//
DELIMITER ;

CALL calculate_interest();
SELECT * FROM TEMPLIST;
```

8 Outputs in Execution

```


MariaDB [(none)]>
MariaDB [(none)]> CREATE database if not exists store;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> use store;
Database changed
MariaDB [store]> CREATE TABLE BOOK (
    ->     Isbn VARCHAR(10) PRIMARY KEY,
    ->     Title VARCHAR(100),
    ->     SoldCopies INT
    -> );
Query OK, 0 rows affected (0.006 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE WRITING (
    ->     Isbn VARCHAR(10),
    ->     Name VARCHAR(50),
    ->     PRIMARY KEY (Isbn, Name),
    ->     FOREIGN KEY (Isbn) REFERENCES BOOK(Isbn)
```

```
    -> );
Query OK, 0 rows affected (0.006 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE AUTHOR (
    ->     Name VARCHAR(50) PRIMARY KEY,
    ->     SoldCopies INT
    -> );
Query OK, 0 rows affected (0.005 sec)

MariaDB [store]>
MariaDB [store]> CREATE TABLE Customer (
    ->     cust_id INT PRIMARY KEY,
    ->     Principal_amount DOUBLE,
    ->     Rate_of_interest DOUBLE,
    ->     Years INT
    -> );
Query OK, 0 rows affected (0.005 sec)

MariaDB [store]>
MariaDB [store]>
MariaDB [store]> INSERT INTO BOOK (Isbn, Title, SoldCopies)
    -> VALUES ('9783161', 'Crime and Punishment', 500);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO WRITING (Isbn, Name)
    -> VALUES ('9783161', 'Fyodor Dostoevsky');
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO AUTHOR (Name, SoldCopies)
    -> VALUES ('Fyodor Dostoevsky', 500);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> INSERT INTO Customer (cust_id, Principal_amount, Rate_of_interest
    , Years)
    -> VALUES (1, 1000, 0.05, 5);
Query OK, 1 row affected (0.001 sec)

MariaDB [store]>
MariaDB [store]> -- Create Triggers
MariaDB [store]>
MariaDB [store]> DELIMITER //
MariaDB [store]> CREATE TRIGGER update_author_soldcopies
    -> AFTER UPDATE ON BOOK
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.SoldCopies != OLD.SoldCopies THEN
    ->         UPDATE AUTHOR
    ->         SET SoldCopies = SoldCopies + (NEW.SoldCopies - OLD.SoldCopies)
    WHERE Name IN (SELECT Name FROM WRITING WHERE Isbn = NEW.Isbn);
    ->     END IF;
    -> END//
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> DELIMITER ;
MariaDB [store]>
```

```
MariaDB [store]> delimiter $$
MariaDB [store]> CREATE TRIGGER insert_author_soldcopies
    -> AFTER INSERT ON WRITING
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE AUTHOR
    ->     SET SoldCopies = SoldCopies + (SELECT SoldCopies FROM BOOK WHERE Isbn =
     NEW.Isbn)
    ->     WHERE Name = NEW.Name;
    -> END $$
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> delimiter ;
MariaDB [store]>
MariaDB [store]> select * from BOOK;
+---------+----------------------+------------+
| Isbn    | Title                | SoldCopies |
+---------+----------------------+------------+
| 9783161 | Crime and Punishment |        500 |
+---------+----------------------+------------+
1 row in set (0.002 sec)

MariaDB [store]> select * from AUTHOR;
+-------------------+------------+
| Name              | SoldCopies |
+-------------------+------------+
| Fyodor Dostoevsky |        500 |
+-------------------+------------+
1 row in set (0.001 sec)

MariaDB [store]> select * from WRITING;
+---------+-------------------+
| Isbn    | Name              |
+---------+-------------------+
| 9783161 | Fyodor Dostoevsky |
+---------+-------------------+
1 row in set (0.001 sec)

MariaDB [store]> select * from Customer;
+---------+------------------+------------------+-------+
| cust_id | Principal_amount | Rate_of_interest | Years |
+---------+------------------+------------------+-------+
|       1 |             1000 |             0.05 |     5 |
+---------+------------------+------------------+-------+
1 row in set (0.001 sec)

MariaDB [store]> USE store;
Database changed
MariaDB [store]>
MariaDB [store]> DROP TABLE IF EXISTS TEMPLIST;
Query OK, 0 rows affected (0.007 sec)

MariaDB [store]> CREATE TABLE TEMPLIST (
    ->   cust_id INT PRIMARY KEY,
    ->   Compound_interest DOUBLE
    -> );
Query OK, 0 rows affected (0.007 sec)

MariaDB [store]>
```

```
MariaDB [store]> DROP PROCEDURE IF EXISTS calculate_interest;
Query OK, 0 rows affected (0.003 sec)

MariaDB [store]> DELIMITER //
MariaDB [store]> CREATE PROCEDURE calculate_interest()
    -> BEGIN
    ->   DECLARE done INT DEFAULT FALSE;
    ->   DECLARE custid INT;
    ->   DECLARE princ_amt, rate, num_years, interest_amt, temp_amt DOUBLE;
    ->   DECLARE cur CURSOR FOR SELECT cust_id, Principal_amount, Rate_of_interest
   , Years FROM Customer;
    ->
    ->   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    ->
    ->   OPEN cur;
    ->   loop1: LOOP
    ->     FETCH cur INTO custid, princ_amt, rate, num_years;
    ->     IF done THEN
    ->       LEAVE loop1;
    ->     END IF;
    ->
    ->     SET temp_amt = 1 + rate / 12;
    ->     SET interest_amt = princ_amt * POWER(temp_amt, num_years * 12) -
   princ_amt;
    ->
    ->     INSERT INTO TEMPLIST (cust_id, Compound_interest) VALUES (custid,
   interest_amt);
    ->   END LOOP;
    ->   CLOSE cur;
    -> END//
Query OK, 0 rows affected (0.002 sec)

MariaDB [store]> DELIMITER ;
MariaDB [store]>
MariaDB [store]> CALL calculate_interest();
Query OK, 1 row affected (0.001 sec)

MariaDB [store]> SELECT * FROM TEMPLIST;
+---------+--------------------+
| cust_id | Compound_interest |
+---------+--------------------+
|       1 | 283.3586785035118 |
+---------+--------------------+
1 row in set (0.000 sec)

MariaDB [store]>
```

9 Conclusion

Thus, we have learned creating and using Cursor in SQL. We have also learned about the different types of Cursors and their advantages and disadvantages.

10 FAQ

1. **Enlist Advantages of Cursors ?**
   Advantages of Cursors:
   (a) Flexibility: Cursors offer a flexible way to process database records one at a time, allowing for complex processing logic to be applied to each record individually.
   (b) Control: Cursors give developers more control over the data being processed, allowing for precise manipulation of records and fields.
   (c) Sequential processing: Cursors allow records to be processed in a sequential order, which can be important in cases where records need to be processed in a specific order.
   (d) Record-level operations: Cursors allow for record-level operations such as inserting, updating, and deleting individual records in a result set.
2. **Enlist Disadvantages of Cursors?**

   Disadvantages of Cursors:
   (a) Overhead: Cursors can add overhead to the database server, as they require resources to be allocated to hold the result set and manage the cursor.
   (b) Performance: Cursors can have a negative impact on database performance, particularly if they are used to process large result sets.
   (c) Complexity: Cursors can make code more complex and difficult to understand, particularly if they are nested or used in conjunction with other database operations.
3. **What are the Applications of Cursors.**

   Applications of Cursors:
   (a) Report generation: Cursors can be used to generate reports based on complex queries that require record-level processing.
   (b) Data validation: Cursors can be used to validate data against complex business rules that cannot be easily expressed in a single query.
   (c) Data migration: Cursors can be used to migrate data between databases, particularly when data needs to be transformed or manipulated during the migration process.
   (d) Batch processing: Cursors can be used for batch processing operations, such as updating a large number of records based on a specific criteria.
4. **Why do we need the Cursors?**

   Cursors are needed in situations where processing of individual records in a result set is required. They provide a way to traverse and manipulate data one record at a time, allowing for complex processing logic to be applied to each record individually. Cursors are particularly useful in scenarios where complex business rules need to be applied to individual records, and in cases where record-level operations such as inserting, updating, and deleting are required. However, cursors can also have a negative impact on database performance and should be used judiciously.