Assignment - 08

AIM: Write PL/SQL Cursor for the given problem statement

Title: Create Cursor using PL/SQL.

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

Theory:

- What is Cursor?

→ A cursor in PL/SQL is a database object that enables row-by-row processing of query results. It act as a pointer to the memory area holding the result set of a SELECT statement, allowing sequential fetching of rows in a procedural block. Cursors are essential for handling multi-row queries where set-based operations are insufficien

- Why do we need Cursor?

Cursors are needed when SQL's declarative nature (e.g. SELECT for sets) cannot handle row-level logic, such as conditional processi ng, calculations per row, or dynamic updates

- Different types of Cursors
  - Implicit Cursors: Automatically created by Oracle/MySQL for single-row DML (e.g, UPDATE, DELETE) No explicit declaration; handle internally.
  - Explicit Cursors: User-defined for multi-row SELECTs. Involves DECLARE, OPEN, FETCH, CLOSE.
    - ↳ Further classified as:
      - Static Cursors
      - Dynamic Cursors
      - Parameterized Cursors
      - REF Cursors.

- Drawbacks of Implicit Cursors
  - Limited Control: No access to row data (NEW/OLD) cannot fetch or loop explicitly.
  - Error-Prone for Multi-Row: Assumes single-row impact; bulk operations may fail silently without checks.
  - No Attributes Access: Cannot use %FOUND, %NOTFOUND for custome logic.
  - Debugging Difficulty: Hidden from user; errors are generic SQL exceptions.

- PL/SQL variables in a Cursor
  ↳ PL/SQL variables are declared in the 'DECLARE' section to hold fetched values.
  Syntax: cust_id_var INT; principal_var DECIMAL(10,2)

  These map to cursor columns via FETCH INTO
  They can be used in calculations or conditionals within the loop.

- OPENing a Cursor:
  ↳ OPEN cursor_name; executes the query, allocates memory for the result set, and positions the cursor before the first row. It prepares the cursor for fetching but does not retrieve data yet. Multiple OPENs on the same cursor require CLOSE first.

- Closing a Cursor:
  ↳ CLOSE cursor_name; releases memory and ~~individ~~ indicates the cursor. Must be called after processing to free resources; attempting FETCH on closed cursor raises INVALID_CURSOR
  Good practice always CLOSE in EXIT handler.

- Cursor attributes:
  - %ISOPEN: TRUE if - cursor open (BOOLEAN).
  - % FOUND: TRUE if last FETCH succeeded (NULL if not OPEN/ FETCHED).
  - %NOTFOUND: TRUE if no more rows
  - %ROWCOUNT : Number of rows fetched so far (INTEGER). Used for loop control:

```
LOOP.... FETCH.... EXIT WHEN %NOTFOUND;
ENDLOOP;
```

Syntax:

```
DELIMITER //
GREATE PRODEEDURE procedure_name()
BEGIN
    -- DECLARE Section
    DECLARE done INT DEFAULT FALSE;
    DECLARE var1 datatype:
    DECLARE var2 datatype;

    --DECLARE CURSOR
    DECLARE cursore_name CURSOR FOR
        SELECT column1, column2 FROM table_name
WHERE condition;

    -- DECLARE HANDLER for end-of-data
    DECLARE CONTINUE HADLER FOR NOT FOUND
SET done = TRUE;
```

```
-- Executable section
OPEN cursore_name;

read_loop: LOOP
    FETCH cursor_name INTO var1, var2;
    IF done THEN
        LEAVE read_loop;
    END. IF;

    -- Process: e.g, calculations, INSERT
    SET calculated_value = var1 * POW (1 + var2
/100, years);
    INSERT INTO templist (cust_id, interest)
VALUES (var1, calculated_value);
END LOOP;

CLOSE cursor_name;
END//
DELIMITER;
```

Input: Cursor for the given problem statement.

Output: Data as per request.

Conclusion: Thus, we have learned creating and using cursor in SQL.

FAQs:

1. Enlist Advantages of Cursors?

→ i) Enable row-by-row processing for complex logic.
ii) Memory-efficient for laorge Dataset.
iii) Reusable via parameters for dynamic queries.
iv) Precise control over iteration and error handling.
v) Integrates well with procedureal code.

2. Enlist Disadvantage of Cursors?

→ i) Performance Hit: Slower than set-based SQL for bulk operations.
ii) Increased code complexity and maintanance.
iii) Resource usage: Open cursors consume memory if not closed.
iv) Error-prone: Manual handling of loops and execeptions.
v) Not scalable for very large results sets.

3. Why do we need the Cursors?

→ i) SQL is set-oriented, but real-world logic often requires sequential, conditional handling (eg: per-customer calculations). Cursor bridge this by allowing imprerative-style programming in the database.

4. What are the Applications of Cursors?

→ i) Generating reports with row-specific Formatting.
ii) Batch processing (e.g: updating records)
iii) Data migration or ETL with transformat
iv) Auditing changes per row
v) Nested processing in hierarchical data.