RDBMS-03-Cursor

Cursor

A cursor is a database object used to retrieve data from a result set returned by a query. It allows for sequential processing of the rows in the result set, enabling manipulation of data row by row.

Cursor Operations

- **Declaration**: Cursors are declared with a `SELECT` statement, specifying the query to be executed.
- **Opening**: Once declared, a cursor needs to be opened to execute the associated query and establish a result set.
- **Fetching**: After opening, rows from the result set can be fetched one at a time or in bulk for processing.
- **Processing**: Fetched data can be processed within loops or other control structures, allowing for various operations such as manipulation, aggregation, or validation.
- **Closing**: After processing, the cursor should be closed to release resources and free up memory.

Common Use

- **Data Manipulation**: Cursors are used to retrieve and modify data row by row, enabling complex data processing tasks.
- **Data Validation**: They are helpful in validating data against certain criteria or constraints.
- **Reporting**: Cursors facilitate generating reports by iterating over result sets and formatting data for display.

Considerations

- **Performance**: Cursors can impact performance, especially when used with large result sets. Efficient query design and optimization techniques should be employed.
- **Resource Management**: Proper cursor management is essential to prevent resource leaks and optimize memory usage.
- **Concurrency**: Cursors can be sensitive to changes made by other transactions. Appropriate isolation levels and locking mechanisms should be considered in multi-user environments.

Implicit Cursor

- Automatically created by the database management system (DBMS) to handle queries.
- They are simple to use but offer limited control over query execution.
- These are created by default by ORACLE itself when DML statements like, `INSERT`, `UPDATE`, and `DELETE` statements are executed.
- They are also created when a `SELECT` statement that returns just one row is executed.

- We cannot use implicit cursors for user defined work.

Explicit Cursor

- Defined explicitly by the programmer using SQL commands.
- They provide more control over the result set, allowing for fine-tuning of data retrieval and manipulation.
- Explicit cursors are user defined cursors written by the developer.
- They can be created when a `SELECT` statement that returns more than one row is executed.
- Even though the cursor stores multiple records, only one record can be processed at a time, which is called as current row. (When you fetch a row, the current row position moves to next row.)

Steps to Manage Explicit Cursor

Declare: A cursor is defined in the declaration section of PL/SQL block

```
DECLARE cursorname CURSOR FOR SELECT ………
```

Open A Cursor: Once cursor is declared we can open it.

- When cursor is opened following operations are performed.
  - Memory is allocated to store the data.
  - Execute `SELECT` statement associated with cursor
  - Create active data set by retrieving data from table
  - Set the cursor row pointer to point to first record in active data set.

```
OPEN cursorname;
```

Fetching Data: We cannot process selected row directly. We have to fetch column values of a row into memory variables. This is done by `FETCH` statement.

```
FETCH NEXT FROM cursorname INTO variable1, variable2………
```

Processing Data: This step involves actual processing of current row.

Closing Cursor: A cursor should be closed after the processing of data completes. Once you close the cursor it will release memory allocated for that cursor.

```
CLOSE cursorname;
```

Deallocating Cursor: It is used to delete a cursor and releases all resources used by cursor.

```
DEALLOCATE cursorname;
```

Cursor Attributes

- Oracle provides some attributes known as Implicit cursor's attributes to check the status of DML operations.

Attributes	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.
%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise, it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

Advantages of Cursor

Control over Data Retrieval:

- Cursors provide precise control over the retrieval of data from a result set.
- They allow you to fetch rows one by one or in batches as needed, which can be beneficial for processing large datasets efficiently.

Iterative Processing:

- Cursors enable iterative processing of result sets, allowing you to perform operations on each row returned by a query. This is useful when you need to apply business logic or calculations to individual records.

Transaction Management:

- Cursors support transaction management features, allowing you to control when data modifications are committed or rolled back.
- This level of control ensures data integrity and consistency, especially in complex transactions involving multiple operations.

Reusable Logic:

- Cursors can encapsulate SQL statements within a PL/SQL block, making the code reusable across different parts of an application. This promotes modularity and code maintainability by centralizing data access logic.

Complex Query Support:

- Cursors enable the execution of complex queries involving joins, subqueries, and aggregations.
- They allow you to process the results of such queries row by row, facilitating data manipulation and analysis.

Scrollable Result Sets:

- Some cursor types, such as explicit cursors with the SCROLL option, provide the ability to navigate through result sets in both forward and backward directions.
- This flexibility is beneficial for implementing features like pagination or browsing

Fine-Grained Error Handling:

- Cursors allow you to handle exceptions and errors at the row level, enabling custom error processing and recovery strategies.
- This level of granularity enhances the robustness and reliability of PL/SQL programs.

Cursor Parameters:

- Cursors can accept parameters, enabling dynamic SQL generation based on runtime conditions.
- This feature allows for more flexible and adaptable code that can cater to varying requirements without hardcoding values.

Performance Optimization:

- Cursors offer performance optimization features such as bulk fetch operations and cursor attributes (like %FOUND, %NOTFOUND, %ROWCOUNT) to minimize round trips between the database and application, thereby improving overall performance.

Data Manipulation:

- Cursors support DML (Data Manipulation Language) statements like INSERT, UPDATE, and DELETE, allowing you to modify data within a result set as needed, either directly or conditionally based on specific criteria.

Disadvantages of Cursor

Performance Overhead:

- Cursors often involve fetching data row by row, which can lead to performance issues, especially when dealing with large datasets.
- This is because each fetch operation incurs additional overhead compared to fetching data in bulk.

Memory Consumption:

- Cursors require memory resources on the database server to store the result set. For large result sets, this can lead to increased memory consumption, potentially causing contention with other processes and impacting overall system performance.

Locking and Blocking:

- Cursors typically hold locks on the rows they are processing until the cursor is closed or the transaction is committed.
- This can lead to locking and blocking issues, especially in scenarios where multiple transactions are accessing the same data concurrently.

Complexity and Maintenance:

- Code written using cursors can be more complex and harder to maintain compared to set-based operations. Cursors require explicit declaration, opening, fetching, and closing, which can make the code harder to understand and debug.

Limited Scalability:

- Cursors can be less scalable compared to set-based operations, especially when dealing with large datasets or high transaction volumes.
- As the size of the data increases, the overhead associated with cursor operations can become prohibitive.

Inefficient Network Usage:

- When cursors are used to fetch data row by row from the database server to the client application, it can result in inefficient network usage, especially over slow or high-latency network connections.

Not Suitable for All Use Cases:

- Cursors are not always the best choice for all use cases. In many scenarios, set-based operations using SQL statements such as SELECT, INSERT, UPDATE, and DELETE can achieve the desired results more efficiently and with better performance.

Examples

Explicit Cursor

```
-- Create a procedure that uses a cursor to fetch data
CREATE OR REPLACE PROCEDURE fetch_users IS

-- Declare variables to hold data fetched by the cursor
v_id users.id % TYPE;
v_name users.name % TYPE;
v_age users.age % TYPE;

-- Declare a cursor
CURSOR user_cursor IS
SELECT
  id, name, age
FROM
  users;

BEGIN

-- Open the cursor
OPEN user_cursor;

-- Fetch data from the cursor into variables
LOOP FETCH user_cursor INTO v_id,
v_name,
v_age;

-- Exit the loop if no more rows to fetch
```

```
EXIT WHEN user_cursor % NOTFOUND;

-- Process the fetched data (here, we'll print it)
DBMS_OUTPUT.PUT_LINE('ID: ' || v_id || ', Name: ' || v_name || ', Age: ' || v_age);

END LOOP;

-- Close the cursor
CLOSE user_cursor;

END;
```

Implicit Cursor

```
DECLARE v_id users.id % TYPE;
v_name users.name % TYPE;
v_age users.age % TYPE;

BEGIN -- Implicit cursor will be used here

FOR user_rec IN
  (SELECT
      id, name, age
  FROM
    users)
LOOP -- Fetch data from the implicit cursor into variables
    v_id := user_rec.id;
    v_name := user_rec.name;
    v_age := user_rec.age;

-- Process the fetched data (here, we'll print it)
DBMS_OUTPUT.PUT_LINE('ID: ' || v_id || ', Name: ' || v_name || ', Age: ' || v_age);
END LOOP;
END;
```

Questions

1. What is cursor? Explain types of cursors.
2. Explain cursor operations.
3. Explain Advantages and Disadvantages of Cursor. (MID-06M)
4. What is Cursor? Explain any three attributes of cursor with description. (MID-03M)
5. Explain an explicit Cursor with syntax

@SOUBCA

Join Us on Telegram @SOU BCA