KOREA TECH
KOREA UNIVERSITY OF TECHNOLOGY & EDUCATION

Database System

Cursors

Muhammad Tariq Mahmood
tariq@koreatech.ac.kr
School of Computer Science and Engineering
Korea University of Technology and Education

Cursors

- A data type for variables or stored procedure OUTPUT parameters that contain a reference to a cursor.
- The cursor is a tool that is used to iterate over a result set, or to loop through each row of a result set one row at a time.
- *Cursor* allows you to process rows from a result set of a query one at a time and in a requested order.
- This is in contrast to using set-based queries

Cursors

- Working with a cursor generally involves the following steps:

1. Declare the cursor based on a query.
2. Open the cursor.
3. Fetch attribute values from the first cursor record into variables.
4. Until the end of the cursor is reached (while the value of a function called *@@FETCH_STATUS* is 0), loop through the cursor records; in each iteration of the loop, fetch attribute values from the current cursor record into variables and perform the processing needed for the current row.
5. Close the cursor.
6. Deallocate the cursor.

Cursors

- Table @Result is declared

```
SET NOCOUNT ON;

DECLARE @Result TABLE
(
custid INT,
ordermonth DATETIME,
qty INT,
runqty INT,
PRIMARY KEY(custid, ordermonth)
);
```

Cursors

- Local variables are declared

```
DECLARE
@custid AS INT,
@prvcustid AS INT,
@ordermonth DATETIME,
@qty AS INT,
@runqty AS INT;
```

Cursors

1. Declare the cursor based on a query.

```
DECLARE C CURSOR FAST_FORWARD  FOR
SELECT custid, ordermonth, qty
FROM Sales.CustOrders
ORDER BY custid, ordermonth;
```

2. Open the cursor.

```
OPEN C;
```

Cursors

3. Fetch attribute values from the first cursor record into variables.

```
FETCH NEXT FROM C INTO @custid, @ordermonth, @qty;
SELECT @prvcustid = @custid, @runqty = 0;
```

- Step-2
  - The cursor C is opened

```
OPEN C;
```

Cursors

4. loop through the cursor records

```
WHILE @@FETCH_STATUS = 0
BEGIN
IF @custid <> @prvcustid
SELECT @prvcustid = @custid, @runqty = 0;
SET @runqty = @runqty + @qty;
INSERT INTO @Result VALUES(@custid, @ordermonth, @qty, @runqty);
FETCH NEXT FROM C INTO @custid, @ordermonth, @qty;
END
```

Cursors

5. Close the cursor.

```
CLOSE C;
```

6. Deallocate the cursor

```
DEALLOCATE C;
```

Cursor Function (@@CURSOR_ROWS)

- @@CURSOR_ROWS : Returns the number of qualifying rows currently in the last cursor opened on the connection

Return value	Description
- m	The cursor is populated asynchronously. The value returned (- m) is the number of rows currently in the keyset.
-1	The cursor is dynamic. Because dynamic cursors reflect all changes, the number of rows that qualify for the cursor is constantly changing. It can never be definitely stated that all qualified rows have been retrieved.
0	No cursors have been opened, no rows qualified for the last opened cursor, or the last-opened cursor is closed or deallocated.
n	The cursor is fully populated. The value returned (n) is the total number of rows in the cursor.

Cursor Function (@@CURSOR_ROWS)

- Example

```
USE AdventureWorks2012;
  GO

  SELECT @@CURSOR_ROWS;

  DECLARE Name_Cursor CURSOR FOR
  SELECT LastName ,@@CURSOR_ROWS FROM Person.Person;
  OPEN Name_Cursor;
  FETCH NEXT FROM Name_Cursor;
  SELECT @@CURSOR_ROWS;
  CLOSE Name_Cursor;
  DEALLOCATE Name_Cursor;
  GO
```

Cursor Function (CURSOR_STATUS)

- CURSOR_STATUS: A scalar function that allows the caller of a stored procedure to determine whether or not the procedure has returned a cursor and result set for a given parameter.

Return value	Cursor name	Cursor variable
1	The result set of the cursor has at least one row.	The cursor allocated to this variable is open. For insensitive and keyset cursors, the result set has at least one row. For dynamic cursors, the result set can have zero, one, or more rows.
0	The result set of the cursor is empty.*	The cursor allocated to this variable is open, but the result set is definitely empty.*
-1	The cursor is closed.	The cursor allocated to this variable is closed.
-2	Not applicable.	Can be: No cursor was assigned to this OUTPUT variable by the previously called procedure. A cursor was assigned to this OUTPUT variable by the previously called procedure, but it was in a closed state upon completion of the procedure. Therefore, the cursor is deallocated and not returned to the calling procedure. There is no cursor assigned to a declared cursor variable.
-3	A cursor with the specified name does not exist.	A cursor variable with the specified name does not exist, or if one exists it has not yet had a cursor allocated to it.

Cursor Function (CURSOR_STATUS)

- Example

```
CREATE TABLE #TMP
  (
     ii int
  )
  GO

  INSERT INTO #TMP(ii) VALUES(1)
  INSERT INTO #TMP(ii) VALUES(2)
  INSERT INTO #TMP(ii) VALUES(3)

  GO

  --Create a cursor.
  DECLARE cur CURSOR
  FOR SELECT * FROM #TMP

  --Display the status of the cursor before and after opening
  --closing the cursor.

  SELECT CURSOR_STATUS('global','cur') AS 'After declare'
  OPEN cur
  SELECT CURSOR_STATUS('global','cur') AS 'After Open'
  CLOSE cur
  SELECT CURSOR_STATUS('global','cur') AS 'After Close'

  --Remove the cursor.
  DEALLOCATE cur

  --Drop the table.
  DROP TABLE #TMP
```

Cursor Functions (@@FETCH_STATUS)

- @@FETCH_STATUS: Returns the status of the last cursor FETCH statement issued against any cursor currently opened by the connection.

Return value	Description
0	The FETCH statement was successful.
-1	The FETCH statement failed or the row was beyond the result set.
-2	The row fetched is missing.

Cursor Functions (@@FETCH_STATUS)

- Example

```
DECLARE Employee_Cursor CURSOR FOR
  SELECT BusinessEntityID, JobTitle
  FROM AdventureWorks2012.HumanResources.Employee;
  OPEN Employee_Cursor;
  FETCH NEXT FROM Employee_Cursor;
  WHILE @@FETCH_STATUS = 0
     BEGIN
        FETCH NEXT FROM Employee_Cursor;
     END;
  CLOSE Employee_Cursor;
  DEALLOCATE Employee_Cursor;
  GO
```

Cursors should not be the first choice

1. First and foremost, when you use cursors you pretty much go against the relational model, which is based on set theory.

2. The record-by-record manipulation done by the cursor has overhead. A certain extra cost is associated with each record manipulation by the cursor when compared to set-based manipulation. Given a set-based query and cursor code that do similar physical processing behind the scenes, the cursor code is usually many times slower than the set-based code.

3. With cursors, you spend a lot of code on the physical aspects of the solution—in other words, on how to process the data (declaring the cursor, opening it, looping through the cursor records, closing the cursor, and deallocating the cursor). With set-based solutions, you mainly focus on the logical aspects of the solution—in other words, on what to get instead of on how to get it. Therefore, cursor solutions tend to be longer, less readable, and harder to maintain compared to set-based solutions.