

Cursors

Database Programming

Cursor

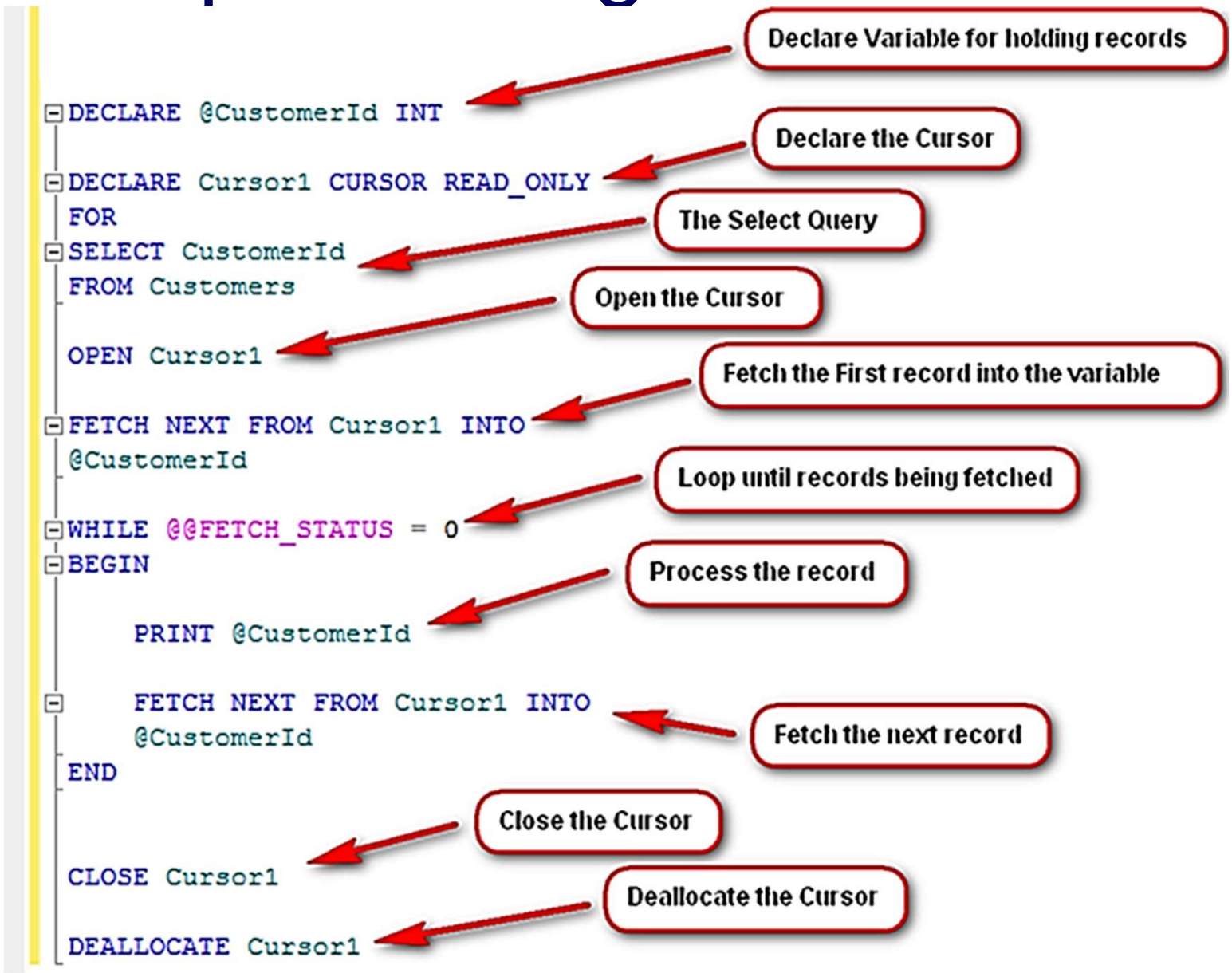
- Is a POINTER data type (handle)
  - Allows you access to one row at a time
  - It is a pointer that will walk through the record set row by row
- A cursor is used to retrieve data from a result set one row at a time
- SQL Server works faster with data as a set than it does with interactive series of commands
  - Cursors are slower and consume more resources
- Cursors should be used as the last option
- Cursors can be faster than a while loop
  - but they do have more overhead

Declaring a Cursor

- Declare Cursor
  - Declares a variable of a CURSOR type
- The declaration includes:
  - The name of the cursor
  - The query used to build the result
  - Its scrolling behavior
  - Scope of variable, updatability, Visibility of updates and deletes
- Basic syntax

```
DECLARE   cursorName     CURSOR
FOR       select_statement
```

Steps in using a Cursor



Steps in using a Cursor (cont.)

- Step 1: Declare the cursor
- Step 2: Open the cursor
- Step 3: Step through the records row by row
  - The FETCH NEXT statement is used
  - @@FETCH_STATUS indicates the success of the FETCH statement
    - @@FETCH_STATUS = 0 => record found
    - @@FETCH_STATUS = -1 => No record found
    - @@FETCH_STATUS = -2 => Deleted record
- Step 4: Close the cursor
  - Releases the current result set associated with the cursor
  - The cursor can be opened again after it is closed
- Step 5: Deallocate (release memory): releases the resources used by the cursor

Example - Using Cursors

```
DECLARE         curEmployeeNames    CURSOR
FOR             SELECT  FirstName , LastName
                FROM    Employee

OPEN                        curEmployeeNames
FETCH NEXT FROM             curEmployeeNames
INTO                        @Fname , @LName

WHILE  @@FETCH_STATUS = 0
BEGIN
        -- Do something with @Fname , @Lname
        SELECT              @Fname , @LName


        FETCH NEXT FROM             curEmployeeNames
        INTO                        @Fname , @LName
END

CLOSE                       curEmployeeNames
DEALLOCATE                  curEmployeeNames
```

Nested Cursors

```
DECLARE          curOuter      CURSOR
FOR              SELECT  FirstName , LastName
                 FROM        Employee

OPEN              curOuter
FETCH NEXT FROM              curOuter
INTO                         @Fname , @LName

WHILE  @@FETCH_STATUS = 0
BEGIN
        DECLARE              curInner      CURSOR
        FOR                  SELECT  C1 , C2          FROM        TTTT
        WHERE                CX = @Fname

        -- Open the INNER cursor
        -- scroll through the rows of the INNER cursor and do something with C1 , C2
        -- Close and deallocate INNER cursor

        FETCH NEXT FROM                curOuter
        INTO                           @Fname , @LName
END

CLOSE                        curOuter
DEALLOCATE                   curOuter
```

Using Cursors - Example 2

- Backup all user defined databases

```
DECLARE     @fileName varchar(100)
DECLARE     @dbName varchar(100)


DECLARE  CurUserDatabases CURSOR
FOR
SELECT name
FROM master.dbo.sysDatabases
WHERE name NOT IN ('master','model','msdb','tempdb')
```

Using Cursors - Example 2 (cont.)

```
OPEN            CurUserDatabases
FETCH NEXT FROM    CurUserDatabases INTO @dbName

WHILE @@FETCH_STATUS = 0
BEGIN
        SET @fileName = 'D:\My_Backup\'
                + CONVERT(varchar(50), GetDate())
                 + @dbName + '.bak'

        BACKUP DATABASE @dbName TO DISK = @filename

        FETCH NEXT FROM CurUserDatabases INTO @dbName
END

CLOSE           CurUserDatabases
DEALLOCATE      CurUserDatabases
```

Example - Using table datatype

```
DECLARE @fileName varchar(100)
DECLARE @dbName varchar(100)
DECLARE @nbDatabases int

SET @fileName =  'D:\My_Backup\' + CONVERT(varchar(50),
        GetDate()) + '.bak'

DECLARE  @tempTable TABLE
         (rowID int IDENTITY ,  name sysname)

INSERT   @tempTable
SELECT   name
FROM     master.dbo.sysDatabases
WHERE    name NOT IN ('master','model','msdb','tempdb')
```

Example - Using table datatype (cont.)

```
SET @nbDatabases  =  @@ROWCOUNT

WHILE ( @nbDatabases > 0)
BEGIN
        SELECT @dbName =
                        ( SELECT name FROM @tempTable
                         WHERE rowID = @nbDatabases )

        BACKUP DATABASE @dbName TO DISK = @filename

        SET @nbDatabases = @nbDatabases - 1
END
```

Example - Using TOP

```
DECLARE @cnt int = 1 --initialize iterator
DECLARE @rowCnt int --rowcount
DECLARE @sValue varchar(100)
DECLARE @mytbl TABLE (Value varchar(100))

INSERT into @mytbl(Value) SELECT name FROM master..sysdatabases

SET @rowCnt = @@ROWCOUNT

WHILE @rowCnt > 0
BEGIN
        SELECT TOP 1 @sValue = Value from @mytbl
        SET @rowCnt = @@ROWCOUNT --ensure that we still have data

        IF @rowCnt > 0
        BEGIN
                print 'My Value is ' + @sValue
                DELETE FROM @mytbl WHERE value = @sValue --processed
        END
END
```

The FETCH Statement

- NEXT
  - Returns the row immediately following the current row
  - If FETCH NEXT is the first fetch against a cursor, it returns the first row in the result set
- PRIOR
  - Returns the row immediately preceding the current row
  - If FETCH PRIOR is the first fetch against a cursor, no row is returned and the cursor is left positioned before the first row.
- FIRST : Returns the first row in the cursor and makes it the current row.
- LAST : Returns the last row in the cursor and makes it the current row.

The FETCH Statement (cont.)

- ABSOLUTE { n | @nvar}
  - If n or @nvar is positive, returns the row n rows from the front of the cursor
- RELATIVE { n | @nvar}
  - If n or @nvar is positive, returns the row n rows beyond the **current** row
  - If n or @nvar is negative, returns the row n rows prior to the **current** row

Types of Cursors

- SQL Server allows you to define four cursor types
- These cursors vary in
  - their ability to detect changes in the base table and
  - in the resources they consume
- A cursor can detect changes to rows only when it attempts to fetch those rows a second time.
- The ability of a cursor to detect changes is also influenced by the transaction isolation level.

Types of Cursors (cont.)

- The four cursor types supported by SQL Server are:
  - Static cursors
  - Dynamic cursors
  - FAST_FORWARD cursors
  - Keyset-driven cursors
- Static cursors detect no changes, and consume relatively few resources while scrolling.
- Dynamic cursors detect all changes, but consume more resources while scrolling.
- FAST_FORWARD cursors are READ_ONLY cursors with performance optimizations enabled
- Keyset-driven cursors lie in between
  - They detect most changes, but with less resource demands than dynamic cursors

Declare Cursors - Syntax

- Declaring a Cursor

```
DECLARE   cursor_name     CURSOR
    [ FORWARD_ONLY | SCROLL ]
    [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]
FOR select_statement
```

- FORWARD_ONLY
  - Can only be scrolled from the first to the last row
  - FETCH NEXT is the only supported fetch option
- SCROLL
  - all FETCH options are supported

Types of Cursors - STATIC

- Defines a cursor that makes a temporary copy of the data
  - Uses tempdb
- Displays the result set as it was when the cursor was opened
- Modifications made by other uses are not seen by the cursor
- Does not display new rows inserted in the database after the cursor was opened
- No UPDATE, INSERT, or DELETE operations are reflected in a static cursor
  - unless the cursor is closed and reopened
- Consume relatively few resources while scrolling

Types of Cursors - KEYSET

- The membership and order of rows in the cursor are fixed
- The set of keys that uniquely identify the rows in the cursor (keyset) is stored into a table
  - The keyset is built in tempdb when the cursor is opened
- Changes to non-key values in the base tables are visible to the cursor
- Inserts made by other users are not visible

Types of Cursors - DYNAMIC

- All data changes made to the rows in its result set are visible to the cursor
  - All UPDATE, INSERT, and DELETE statements made by all users are visible through the cursor.
- Data values, order, and membership of the rows can change on each fetch
- DYNAMIC SCROLL cursors support all the FETCH options except ABSOLUTE

Types of Cursors - FAST_FORWARD

- The default type of cursor
- A FORWARD_ONLY, READ_ONLY cursor with performance optimizations
- FAST_FORWARD CAN NOT be specified if SCROLL is also specified.
- FAST_FORWARD cursors produce the least amount of overhead on SQL Server

Types of Cursors - Example 1

```
DECLARE @firstName as NVARCHAR(50);
DECLARE @lastName as NVARCHAR(50);
DECLARE @salary as DECIMAL(8,2);
DECLARE @EmpSalaryCursor as CURSOR;

SET @EmpSalaryCursor = CURSOR FORWARD_ONLY FOR
SELECT TOP 10 FName, LName, Salary
FROM Employee

OPEN @EmpSalaryCursor;
FETCH NEXT FROM @EmpSalaryCursor INTO @firstName, @lastName, @salary;
WHILE @@FETCH_STATUS = 0
BEGIN
        PRINT ISNULL(@firstName, '') + ' ‘ + ISNULL(@lastName, '')  +
            ' Salary: ' + ISNULL(CAST(@salary AS Varchar), '') ;
        FETCH NEXT FROM @EmpSalaryCursor INTO @firstName, @lastName, @salary
END
```

Types of Cursors - Example 2

- Using scrollable cursors

```
DECLARE       cEmployee CURSOR SCROLL
FOR     SELECT FirstName FROM Employee

OPEN cEmployee

FETCH FIRST           FROM cEmployee INTO @FName
FETCH LAST            FROM cEmployee INTO @FName
FETCH ABSOLUTE 2      FROM cEmployee INTO @FName
FETCH RELATIVE -2     FROM cEmployee INTO @FName

SELECT @FName
```