**Cursors**

**This file contains little parts of the SQL SERVER Help system using the index for cursors.**

**Transact-SQL cursors**

Are based on the DECLARE CURSOR syntax and are used mainly in Transact-SQL scripts, stored procedures, and triggers. Transact-SQL cursors are implemented on the server and are managed by Transact-SQL statements sent from the client to the server. They may also be contained in batches, stored procedures, or triggers.

From cursors [SQL SERVER] then attributes :

Topic link iconTransact-SQL Syntax Conventions

**ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de_6tsql/local/collapse.gifSyntax**

|  |
| --- |
|  |
| **ISO Syntax**  DECLARE *cursor\_name* [ INSENSITIVE ] [ SCROLL ] CURSOR  FOR *select\_statement*  [ FOR { READ ONLY | UPDATE [ OF *column\_name* [ **,**...n ] ] } ]  [;]  **Transact-SQL Extended Syntax**  DECLARE *cursor\_name* CURSOR [ LOCAL | GLOBAL ]  [ FORWARD\_ONLY | SCROLL ]  [ STATIC | KEYSET | DYNAMIC | FAST\_FORWARD ]  [ READ\_ONLY | SCROLL\_LOCKS | OPTIMISTIC ]  [ TYPE\_WARNING ]  FOR *select\_statement*  [ FOR UPDATE [ OF *column\_name* [ **,**...n ] ] ]  [;] |

### A. Using simple cursor and syntax

The result set generated at the opening of this cursor includes all rows and all columns in the table. This cursor can be updated, and all updates and deletes are represented in fetches made against this cursor. FETCH NEXT is the only fetch available because the SCROLL option has not been specified.

|  |  |
| --- | --- |
|  | ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de_6tsql/local/copycode.gifCopy Code |
| DECLARE vend\_cursor CURSOR      FOR SELECT \* FROM Purchasing.Vendor  OPEN vend\_cursor  FETCH NEXT FROM vend\_cursor | |

### A. Using FETCH in a simple cursor

The following example declares a simple cursor for the rows in the Person.Contact table with a last name that starts with B, and uses FETCH NEXT to step through the rows. The FETCH statements return the value for the column specified in DECLARE CURSOR as a single-row result set.

|  |  |
| --- | --- |
|  | ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de_6tsql/local/copycode.gifCopy Code |
| USE AdventureWorks  GO  DECLARE contact\_cursor CURSOR FOR  SELECT LastName FROM Person.Contact  WHERE LastName LIKE 'B%'  ORDER BY LastName  OPEN contact\_cursor  -- Perform the first fetch.  FETCH NEXT FROM contact\_cursor  -- Check @@FETCH\_STATUS to see if there are any more rows to fetch.  WHILE @@FETCH\_STATUS = 0  BEGIN  -- This is executed as long as the previous fetch succeeds.  FETCH NEXT FROM contact\_cursor  END  CLOSE contact\_cursor  DEALLOCATE contact\_cursor  GO | |

### B. Using FETCH to store values in variables

The following example is similar to example A, except the output of the FETCH statements is stored in local variables instead of being returned directly to the client. The PRINT statement combines the variables into a single string and returns them to the client.

|  |  |
| --- | --- |
|  | ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de_6tsql/local/copycode.gifCopy Code |
| USE AdventureWorks  GO  -- Declare the variables to store the values returned by FETCH.  DECLARE @LastName varchar(50), @FirstName varchar(50)  DECLARE contact\_cursor CURSOR FOR  SELECT LastName, FirstName FROM Person.Contact  WHERE LastName LIKE 'B%'  ORDER BY LastName, FirstName  OPEN contact\_cursor  -- Perform the first fetch and store the values in variables.  -- Note: The variables are in the same order as the columns  -- in the SELECT statement.  FETCH NEXT FROM contact\_cursor  INTO @LastName, @FirstName  -- Check @@FETCH\_STATUS to see if there are any more rows to fetch.  WHILE @@FETCH\_STATUS = 0  BEGIN  -- Concatenate and display the current values in the variables.  PRINT 'Contact Name: ' + @FirstName + ' ' + @LastName  -- This is executed as long as the previous fetch succeeds.  FETCH NEXT FROM contact\_cursor  INTO @LastName, @FirstName  END  CLOSE contact\_cursor  DEALLOCATE contact\_cursor  GO | |

### Using nested cursors to produce report output

The following example shows how cursors can be nested to produce complex reports. The inner cursor is declared for each vendor.

|  |  |
| --- | --- |
|  | ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de_6tsql/local/copycode.gifCopy Code |
| SET NOCOUNT ON  DECLARE @vendor\_id int, @vendor\_name nvarchar(50),      @message varchar(80), @product nvarchar(50)  PRINT '-------- Vendor Products Report --------'  DECLARE vendor\_cursor CURSOR FOR  SELECT VendorID, Name  FROM Purchasing.Vendor  WHERE PreferredVendorStatus = 1  ORDER BY VendorID  OPEN vendor\_cursor  FETCH NEXT FROM vendor\_cursor  INTO @vendor\_id, @vendor\_name  WHILE @@FETCH\_STATUS = 0  BEGIN      PRINT ' '      SELECT @message = '----- Products From Vendor: ' +          @vendor\_name      PRINT @message      -- Declare an inner cursor based      -- on vendor\_id from the outer cursor.      DECLARE product\_cursor CURSOR FOR      SELECT v.Name      FROM Purchasing.ProductVendor pv, Production.Product v      WHERE pv.ProductID = v.ProductID AND      pv.VendorID = @vendor\_id  -- Variable value from the outer cursor      OPEN product\_cursor      FETCH NEXT FROM product\_cursor INTO @product      IF @@FETCH\_STATUS <> 0          PRINT ' <<None>>'      WHILE @@FETCH\_STATUS = 0      BEGIN          SELECT @message = ' ' + @product          PRINT @message          FETCH NEXT FROM product\_cursor INTO @product          END      CLOSE product\_cursor      DEALLOCATE product\_cursor          -- Get the next vendor.      FETCH NEXT FROM vendor\_cursor      INTO @vendor\_id, @vendor\_name  END  CLOSE vendor\_cursor  DEALLOCATE vendor\_cursor | |

INTO **@***variable\_name*[ **,***...n*]

Allows data from the columns of a fetch to be placed into local variables. Each variable in the list, from left to right, is associated with the corresponding column in the cursor result set. The data type of each variable must either match or be a supported implicit conversion of the data type of the corresponding result set column. The number of variables must match the number of columns in the cursor select list.

ms-help://MS.SQLCC.v10/MS.SQLSVR.v10.en/s10de_1devconc/local/collapse.gifServer Cursors vs. Default Result Sets

Using a cursor is less efficient than using a default result set. In a default result set the only packet sent from the client to the server is the packet containing the statement to execute. When using a server cursor, each FETCH statement must be sent from the client to the server, where it must be parsed and compiled into an execution plan.

If a Transact-SQL statement will return a relatively small result set that can be cached in the memory available to the client application, and you know before executing the statement that you must retrieve the entire result set, use a default result set. Use server cursors **only when** cursor operations **are required to support** the functionality of the application, or when only part of the result set is likely to be retrieved.

A potential drawback of server cursors is that they currently do not support all Transact-SQL statements. Server cursors do not support Transact-SQL statements that generate multiple result sets; therefore, they cannot be used when the application executes a stored procedure or a batch that contain more than one SELECT statement. Server cursors also do not support SQL statements containing the keywords COMPUTE, COMPUTE BY, FOR BROWSE, or INTO.