

Cursor is a database object to retrieve data from a result set one row at a time, instead of the T-SQL commands that operate on all the rows in the result set at one time. We use a cursor when we need to update records in a database table in singleton fashion means row by row.

## Life Cycle of Cursor

### 1. Declare Cursor

A cursor is declared by defining the SQL statement that returns a result set.

### 2. Open

A Cursor is opened and populated by executing the SQL statement defined by the cursor.

### 3. Fetch

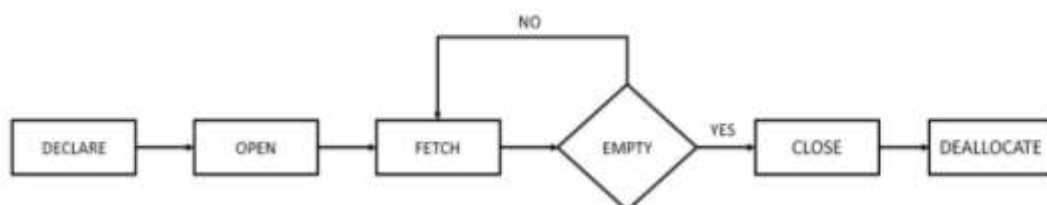
When the cursor is opened, rows can be fetched from the cursor one by one or in a block to do data manipulation.

### 4. Close

After data manipulation, we should close the cursor explicitly.

### 5. Deallocate

Finally, we need to delete the cursor definition and released all the system resources associated with the cursor.



## Syntax to Declare Cursor

Declare Cursor SQL Command is used to define the cursor with many options that impact the scalability and loading behavior of the cursor. The basic syntax is given below

```
DECLARE cursor_name CURSOR

[STATIC | KEYSET | DYNAMIC | FAST_FORWARD] --basic type of cursor

[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC] --define locks

FOR select_statement --define SQL Select statement
```

## Syntax to Open Cursor

A Cursor can be opened locally or globally. By default, it is opened locally. The basic syntax to open cursor is given below:

```
OPEN cursor_name --by default it is local
```

## Syntax to Fetch Cursor

Fetch statement provides the many options to retrieve the rows from the cursor. NEXT is the default option. The basic syntax to fetch cursor is given below:

```
FETCH [NEXT|FIRST|LAST]

FROM [GLOBAL] cursor_name

INTO @Variable_name[1,2,..n]
```

## Syntax to Close Cursor

The close statement closed the cursor explicitly. The basic syntax to close cursor is given below:

```
CLOSE cursor_name --after closing it can be reopen
```

## Syntax to Deallocate Cursor

Deallocate statement delete the cursor definition and free all the system resources associated with the cursor. The basic syntax to close cursor is given below:

```
DEALLOCATE cursor_name --after deallocation it can't be reopen
```

# Example

```
CREATE TABLE Employee

(

    EmpID int PRIMARY KEY,

    EmpName varchar (50) NOT NULL,

    Salary int NOT NULL,

    Address varchar (200) NOT NULL,

)

GO

INSERT INTO Employee(EmpID,EmpName,Salary,Address) VALUES(1,'Mohan',12000,'Noida')

INSERT INTO Employee(EmpID,EmpName,Salary,Address) VALUES(2,'Pavan',25000,'Delhi')

INSERT INTO Employee(EmpID,EmpName,Salary,Address) VALUES(3,'Amit',22000,'Dehradun')

INSERT INTO Employee(EmpID,EmpName,Salary,Address) VALUES(4,'Sonu',22000,'Noida')

INSERT INTO Employee(EmpID,EmpName,Salary,Address) VALUES(5,'Deepak',28000,'Gurgaon')

GO

SELECT * FROM Employee
```

Results		Messages		
	EmpID	Name	Salary	Address
1	1	Mohan	12000	Noida
2	2	Pavan	25000	Delhi
3	3	Amit	22000	Dehradun
4	4	Sonu	22000	Noida
5	5	Deepak	28000	Gurgaon

```
SET NOCOUNT ON

DECLARE @Id int

DECLARE @name varchar(50)

DECLARE @salary int

DECLARE cur_emp CURSOR
```

```

STATIC FOR

SELECT EmpID,EmpName,Salary from Employee

OPEN cur_emp

IF @@CURSOR_ROWS > 0

BEGIN

    FETCH NEXT FROM cur_emp INTO @Id,@name,@salary

    WHILE @@Fetch_status = 0

    BEGIN

        PRINT 'ID : '+ convert(varchar(20),@Id)+' , Name : '+@name+ ' , Salary : '+convert(
varchar(20),@salary)

        FETCH NEXT FROM cur_emp INTO @Id,@name,@salary

    END

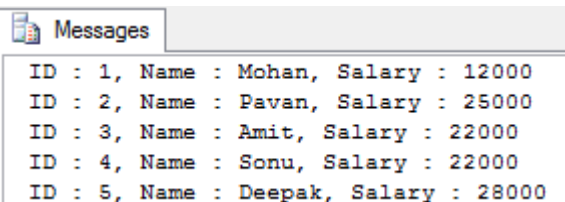
END

CLOSE cur_emp

DEALLOCATE cur_emp

SET NOCOUNT OFF

```



Messages

```

ID : 1, Name : Mohan, Salary : 12000
ID : 2, Name : Pavan, Salary : 25000
ID : 3, Name : Amit, Salary : 22000
ID : 4, Name : Sonu, Salary : 22000
ID : 5, Name : Deepak, Salary : 28000

```

A Cursor allow us to retrieve data from a result set in singleton fashion means row by row. Cursor are required when we need to update records in a database table one row at a time.

A Cursor impacts the performance of the SQL Server since it uses the SQL Server instances' memory, reduce concurrency, decrease network bandwidth and lock resources. Hence it is mandatory to understand the cursor types and its functions so that you can use suitable cursor according to your needs.

# Types of Cursors

## 1. Static Cursors

A static cursor populates the result set at the time of cursor creation and the query result is cached for the lifetime of the cursor. A static cursor can move forward and backward direction. A static cursor is slower and use more memory in comparison to other cursor. Hence you should use it only if scrolling is required and other types of cursors are not suitable.

No UPDATE, INSERT, or DELETE operations are reflected in a static cursor (unless the cursor is closed and reopened). By default static cursors are scrollable. SQL Server static cursors are always read-only.

## 2. Dynamic Cursors

A dynamic cursor allows you to see the data updation, deletion and insertion in the data source while the cursor is open. Hence a dynamic cursor is sensitive to any changes to the data source and supports update, delete operations. By default dynamic cursors are scrollable.

## 3. Forward Only Cursors

A forward only cursor is the fastest cursor among the all cursors but it doesn't support backward scrolling. You can update, delete data using Forward Only cursor. It is sensitive to any changes to the original data source.

There are three more types of Forward Only Cursors. Forward\_Only KEYSET, FORWARD\_ONLY STATIC and FAST\_FORWARD.

A **FORWARD\_ONLY STATIC Cursor** is populated at the time of creation and cached the data to the cursor lifetime. It is not sensitive to any changes to the data source.

A **FAST\_FORWARD Cursor** is the fastest cursor and it is not sensitive to any changes to the data source.

## 4. Keyset Driven Cursors

A keyset driven cursor is controlled by a set of unique identifiers as the keys in the keyset. The keyset depends on all the rows that qualified the SELECT statement at the time of the cursor was opened. A keyset driven cursor is sensitive to any changes to the data source and supports update, delete operations. By default keyset driven cursors are scrollable.

# Static Cursor - Example

```
SET NOCOUNT ON

DECLARE @Id int

DECLARE @name varchar(50)

DECLARE @salary int

DECLARE cur_emp CURSOR

STATIC FOR

SELECT EmpID,EmpName,Salary from Employee

OPEN cur_emp

IF @@CURSOR_ROWS > 0

BEGIN

    FETCH NEXT FROM cur_emp INTO @Id,@name,@salary

    WHILE @@Fetch_status = 0

    BEGIN

        PRINT 'ID : ' + convert(varchar(20),@Id)+', Name : '+@name+ ', Salary : '+convert(
varchar(20),@salary)

        FETCH NEXT FROM cur_emp INTO @Id,@name,@salary

    END

END

CLOSE cur_emp

DEALLOCATE cur_emp

SET NOCOUNT OFF
```



## Messages

```
ID : 1, Name : Mohan, Salary : 12000
ID : 2, Name : Pavan, Salary : 25000
ID : 3, Name : Amit, Salary : 22000
ID : 4, Name : Sonu, Salary : 22000
ID : 5, Name : Deepak, Salary : 28000
```

# Dynamic Cursor - Example

```
--Dynamic Cursor for Update

SET NOCOUNT ON

DECLARE @Id int

DECLARE @name varchar(50)

    DECLARE Dynamic_cur_empupdate CURSOR

DYNAMIC

FOR

SELECT EmpID,EmpName from Employee ORDER BY EmpName

OPEN Dynamic_cur_empupdate

IF @@CURSOR_ROWS > 0

    BEGIN

        FETCH NEXT FROM Dynamic_cur_empupdate INTO @Id,@name

        WHILE @@Fetch_status = 0

            BEGIN

                IF @name='Mohan'

                    Update Employee SET Salary=15000 WHERE CURRENT OF Dynamic_cur_empupdate

                FETCH NEXT FROM Dynamic_cur_empupdate INTO @Id,@name

            END

        END

    END

CLOSE Dynamic_cur_empupdate

DEALLOCATE Dynamic_cur_empupdate

SET NOCOUNT OFF

Go

Select * from Employee
```

Results		Messages		
	EmpID	EmpName	Salary	Address
1	1	Mohan	15000	Noida
2	2	Pavan	25000	Delhi
3	3	Amit	22000	Dehradun
4	4	Sonu	22000	Noida
5	5	Deepak	28000	Gurgaon

```
-- Dynamic Cursor for DELETE

SET NOCOUNT ON

DECLARE @Id int

DECLARE @name varchar(50)

DECLARE Dynamic_cur_empdelete CURSOR

DYNAMIC

FOR

SELECT EmpID,EmpName from Employee ORDER BY EmpName

OPEN Dynamic_cur_empdelete

IF @@CURSOR_ROWS > 0

BEGIN

    FETCH NEXT FROM Dynamic_cur_empdelete INTO @Id,@name

    WHILE @@Fetch_status = 0

    BEGIN

        IF @name='Deepak'

        DELETE Employee WHERE CURRENT OF Dynamic_cur_empdelete

        FETCH NEXT FROM Dynamic_cur_empdelete INTO @Id,@name

    END

END

CLOSE Dynamic_cur_empdelete

DEALLOCATE Dynamic_cur_empdelete

SET NOCOUNT OFF
```



Go

Select \* from Employee

Results		Messages		
	EmpID	EmpName	Salary	Address
1	1	Mohan	15000	Noida
2	2	Pavan	25000	Delhi
3	3	Amit	22000	Dehradun
4	4	Sonu	22000	Noida

## Forward Only Cursor - Example

```
--Forward Only Cursor for Update

SET NOCOUNT ON

DECLARE @Id int

DECLARE @name varchar(50)

DECLARE Forward_cur_empupdate CURSOR

FORWARD_ONLY

FOR

SELECT EmpID,EmpName from Employee ORDER BY EmpName

OPEN Forward_cur_empupdate

IF @@CURSOR_ROWS > 0

BEGIN

    FETCH NEXT FROM Forward_cur_empupdate INTO @Id,@name

    WHILE @@Fetch_status = 0

    BEGIN

        IF @name='Amit'

            Update Employee SET Salary=24000 WHERE CURRENT OF Forward_cur_empupdate

        FETCH NEXT FROM Forward_cur_empupdate INTO @Id,@name

    END

END

END
```

```

CLOSE Forward_cur_empupdate

DEALLOCATE Forward_cur_empupdate

SET NOCOUNT OFF

Go

Select * from Employee

```

Results		Messages		
	EmpID	EmpName	Salary	Address
1	1	Mohan	15000	Noida
2	2	Pavan	25000	Delhi
3	3	Amit	24000	Dehradun
4	4	Sonu	22000	Noida

```

-- Forward Only Cursor for Delete

SET NOCOUNT ON

DECLARE @Id int

DECLARE @name varchar(50)

DECLARE Forward_cur_empdelete CURSOR

FORWARD_ONLY

FOR

SELECT EmpID,EmpName from Employee ORDER BY EmpName

OPEN Forward_cur_empdelete

IF @@CURSOR_ROWS > 0

BEGIN

FETCH NEXT FROM Forward_cur_empdelete INTO @Id,@name

WHILE @@Fetch_status = 0

BEGIN

IF @name='Sonu'

DELETE Employee WHERE CURRENT OF Forward_cur_empdelete

FETCH NEXT FROM Forward_cur_empdelete INTO @Id,@name

```

```

END

END

CLOSE Forward_cur_empdelete

DEALLOCATE Forward_cur_empdelete

SET NOCOUNT OFF

Go

Select * from Employee

```

Results		Messages		
	EmpID	EmpName	Salary	Address
1	1	Mohan	15000	Noida
2	2	Pavan	25000	Delhi
3	3	Amit	24000	Dehradun

## Keyset Driven Cursor - Example

```

-- Keyset driven Cursor for Update

SET NOCOUNT ON

DECLARE @Id int

DECLARE @name varchar(50)

DECLARE Keyset_cur_empupdate CURSOR

KEYSET

FOR

SELECT EmpID,EmpName from Employee ORDER BY EmpName

OPEN Keyset_cur_empupdate

IF @@CURSOR_ROWS > 0

BEGIN

    FETCH NEXT FROM Keyset_cur_empupdate INTO @Id,@name

    WHILE @@Fetch_status = 0

    BEGIN

```

```

IF @name='Pavan'

Update Employee SET Salary=27000 WHERE CURRENT OF Keyset_cur_empupdate

FETCH NEXT FROM Keyset_cur_empupdate INTO @Id,@name

END

END

CLOSE Keyset_cur_empupdate

DEALLOCATE Keyset_cur_empupdate

SET NOCOUNT OFF

Go

Select * from Employee

```

Results		Messages		
	EmpID	EmpName	Salary	Address
1	1	Mohan	15000	Noida
2	2	Pavan	27000	Delhi
3	3	Amit	24000	Dehradun

```

-- Keyse Driven Cursor for Delete

SET NOCOUNT ON

DECLARE @Id int

DECLARE @name varchar(50)

DECLARE Keyset_cur_empdelete CURSOR

KEYSET

FOR

SELECT EmpID,EmpName from Employee ORDER BY EmpName

OPEN Keyset_cur_empdelete

IF @@CURSOR_ROWS > 0

BEGIN

FETCH NEXT FROM Keyset_cur_empdelete INTO @Id,@name

WHILE @@Fetch_status = 0

```

```

BEGIN

IF @name='Amit'

DELETE Employee WHERE CURRENT OF Keyset_cur_empdelete

FETCH NEXT FROM Keyset_cur_empdelete INTO @Id,@name

END

END

CLOSE Keyset_cur_empdelete

DEALLOCATE Keyset_cur_empdelete

SET NOCOUNT OFF

Go Select * from Employee

```

Results		Messages		
	EmpID	EmpName	Salary	Address
1	1	Mohan	15000	Noida
2	2	Pavan	27000	Delhi

## Exercise

Use the production.products table from the [sample database](#)

<b>production.products</b>
* product_id
product_name
brand_id
category_id
model_year
list_price

First, declare two variables to hold product name and list price, and a cursor to hold the result of a query that retrieves product name and list price from the production.products table.