Cursores en Transact SQL

Un cursor es una variable que nos permite recorrer con un conjunto de resultados obtenido a través de una sentencia SELECT fila a fila.

Cuando trabajemos con cursores debemos seguir los siguientes pasos.

- Declarar el cursor, utilizando **DECLARE**
- Abrir el cursor, utilizando **OPEN**
- Leer los datos del cursor, utilizando **FETCH** ... **INTO**
- Cerrar el cursor, utilizando **CLOSE**
- Liberar el cursor, utilizando **DEALLOCATE**

La sintaxis general para trabajar con un cursor es la siguiente.

```
-- Declaración del cursor
DECLARE <nombre_cursor> CURSOR
FOR
<sentencia_sql>

-- apertura del cursor
OPEN <nombre_cursor>

-- Lectura de la primera fila del cursor
FETCH <nombre_cursor> INTO <lista_variables>

WHILE (@@FETCH_STATUS = 0)
BEGIN

        -- Lectura de la siguiente fila de un cursor

        FETCH <nombre_cursor> INTO <lista_variables>

        ...

END -- Fin del bucle WHILE

-- Cierra el cursor
CLOSE <nombre_cursor>
-- Libera los recursos del cursor
DEALLOCATE <nombre_cursor>
```

El siguente ejemplo muestra el uso de un cursor.

```
-- Declaracion de variables para el cursor
DECLARE @Id int,
   @Nombre varchar(255),
   @Apellido1 varchar(255),
   @Apellido2 varchar(255),
   @NifCif varchar(20),
   @FxNacimiento datetime

-- Declaración del cursor
DECLARE cClientes CURSOR FOR
SELECT  Id, Nombre, Apellido1,
```

```
    Apellido2, NifCif, FxNacimiento
FROM CLIENTES
-- Apertura del cursor
OPEN cClientes
-- Lectura de la primera fila del cursor
FETCH cClientes INTO    @id, @Nombre, @Apellido1,
                        @Apellido2, @NifCif, @FxNacimiento

WHILE (@@FETCH_STATUS = 0 )
BEGIN

        PRINT @Nombre + ' ' + @Apellido1 + ' ' + @Apellido2

        -- Lectura de la siguiente fila del cursor

        FETCH cClientes INTO    @id, @Nombre, @Apellido1,
                                @Apellido2, @NifCif, @FxNacimiento

END

-- Cierre del cursor
CLOSE cClientes
-- Liberar los recursos
DEALLOCATE cClientes
```

Cuando trabajamos con cursores, la funcion @@FETCH_STATUS nos indica el estado de la última instrucción FETCH emitida, los valores posibles son:

Valor devuelto	Descripción
0	La instrucción FETCH se ejecutó correctamente.
-1	La instrucción FETCH no se ejecutó correctamente o la fila estaba más allá del conjunto de resultados.
-2	Falta la fila recuperada.

En la apertura del cursor, podemos especificar los siguientes parámetros:

```
DECLARE <nombre_cursor> CURSOR

        [ LOCAL | GLOBAL ]

        [ FORWARD_ONLY | SCROLL ]

        [ STATIC | KEYSET | DYNAMIC | FAST_FORWARD ]

        [ READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]

        [ TYPE_WARNING ]

FOR <sentencia_sql>
```

El primer conjunto de parámetros que podemos especificar es [LOCAL | GLOBAL]. A continuación mostramos el significado de cada una de estas opciones.

- **LOCAL**
  Especifica que el ámbito del cursor es local para el proceso por lotes, procedimiento

almacenado o desencadenador en que se creó el cursor.

```
DECLARE cClientes CURSOR LOCAL FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

- **GLOBAL**
  Especifica que el ámbito del cursor es global para la conexión. Puede hacerse referencia al nombre del cursor en cualquier procedimiento almacenado o proceso por lotes que se ejecute en la conexión.

```
DECLARE cClientes CURSOR GLOBAL FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

Si no se especifica GLOBAL ni LOCAL, el valor predeterminado se controla mediante la configuración de la opción de base de datos default to local cursor.

El siguiente conjunto de parámetros que podemos especificar es [FORWARD_ONLY | SCROLL]. A continuación mostramos el significado de cada una de estas opciones.

- **FORWARD_ONLY**
  Especifica que el cursor sólo se puede desplazar de la primera a la última fila. FETCH NEXT es la única opción de recuperación admitida.

```
DECLARE cClientes CURSOR FORWARD_ONLY FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

- **SCROLL**
  Especifica que están disponibles todas las opciones de recuperación (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Si no se especifica SCROLL en una instrucción DECLARE CURSOR la única opción de recuperación que se admite es NEXT. No es posible especificar SCROLL si se incluye también FAST_FORWARD.
  Si se incluye la opción SCROLL, la forma en la que realizamos la lectura del cursor varia, debiendo utilizar la siguiente sintaxis: FETCH [NEXT | PRIOR | FIRST | LAST | RELATIVE | ABSOLUTE] FROM < INTO

```
-- Declaracion de variables para el cursor
DECLARE @Id int,
  @Nombre varchar(255),
  @Apellido1 varchar(255),
  @Apellido2 varchar(255),
  @NifCif varchar(20),
  @FxNacimiento datetime

-- Declaración del cursor
DECLARE cClientes CURSOR SCROLL FOR
SELECT  Id, Nombre, Apellido1,
```

```
    Apellido2, NifCif, FxNacimiento
FROM CLIENTES
-- Apertura del cursor
OPEN cClientes
-- Lectura de la primera fila del cursor
FETCH NEXT FROM cClientes
INTO @id, @Nombre, @Apellido1, @Apellido2, @NifCif, @FxNacimiento

WHILE (@@FETCH_STATUS = 0 )
BEGIN

        PRINT @Nombre + ' ' + @Apellido1 + ' ' + @Apellido2

        -- Lectura de la siguiente fila del cursor

        FETCH NEXT FROM cClientes

        INTO @id,@Nombre,@Apellido1,@Apellido2,@NifCif,@FxNacimiento

END
-- Lectura de la fila anterior
FETCH PRIOR FROM cClientes
INTO @id, @Nombre, @Apellido1, @Apellido2, @NifCif, @FxNacimiento
PRINT @Nombre + ' ' + @Apellido1 + ' ' + @Apellido2
-- Cierre del cursor
CLOSE cClientes
-- Liberar los recursos
DEALLOCATE cClientes
```

El siguiente conjunto de parámetros que podemos especificar es [STATIC | KEYSET | DYNAMIC | FAST_FORWARD]. A continuación mostramos el significado de cada una de estas opciones.

- **STATIC**
  Define un cursor que hace una copia temporal de los datos que va a utilizar. Todas las solicitudes que se realizan al cursor se responden desde esta tabla temporal de tempdb; por tanto, las modificaciones realizadas en las tablas base no se reflejan en los datos devueltos por las operaciones de recuperación realizadas en el cursor y además este cursor no admite modificaciones.

```
DECLARE cClientes CURSOR STATIC FOR
SELECT  Id, Nombre, Apellido1,
    Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

- **KEYSET**
  Especifica que la pertenencia y el orden de las filas del cursor se fijan cuando se abre el cursor. El conjunto de claves que identifica las filas de forma única está integrado en la tabla denominada keyset de tempdb.

```
DECLARE cClientes CURSOR KEYSET FOR
SELECT  Id, Nombre, Apellido1,
    Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

- **DYNAMIC**
  Define un cursor que, al desplazarse por él, refleja en su conjunto de resultados todos los cambios realizados en los datos de las filas. Los valores de los datos, el orden y la pertenencia de las filas pueden cambiar en cada operación de recuperación. La opción de recuperación ABSOLUTE no se puede utilizar en los cursores dinámicos.

```
DECLARE cClientes CURSOR DYNAMIC FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

- **FAST_FORWARD**
  Especifica un cursor FORWARD_ONLY, READ_ONLY con las optimizaciones de rendimiento habilitadas. No se puede especificar FAST_FORWARD si se especifica también SCROLL o FOR_UPDATE.

```
DECLARE cClientes CURSOR FAST_FORWARD FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

En SQL Server 2000, las opciones de cursor FAST_FORWARD y FORWARD_ONLY se excluyen mutuamente. Si se especifican ambas, se genera un error. En SQL Server 2005, las dos palabras clave se pueden utilizar en la misma instrucción DECLARE CURSOR.

El siguiente conjunto de parámetros que podemos especificar es [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]. A continuación mostramos el significado de cada una de estas opciones.

- **READ_ONLY**
  Evita que se efectúen actualizaciones a través de este cursor. No es posible hacer referencia al cursor en una cláusula WHERE CURRENT OF de una instrucción UPDATE o DELETE. Esta opción reemplaza la capacidad de actualizar el cursor.

```
DECLARE cClientes CURSOR READ_ONLY FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

- **SCROLL_LOCKS**
  Especifica que se garantiza que las actualizaciones o eliminaciones posicionadas realizadas a través del cursor serán correctas. Microsoft SQL Server bloquea las filas cuando se leen en el cursor para garantizar que estarán disponibles para futuras modificaciones. No es posible especificar SCROLL_LOCKS si se especifica también FAST_FORWARD o STATIC.

```
DECLARE cClientes CURSOR SCROLL_LOCKS FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

- **OPTIMISTIC**
  Especifica que las actualizaciones o eliminaciones posicionadas realizadas a través del cursor no se realizarán correctamente si la fila se ha actualizado después de ser leída en el cursor. SQL

Server no bloquea las filas al leerlas en el cursor. En su lugar, utiliza comparaciones de valores de columna timestamp o un valor de suma de comprobación si la tabla no tiene columnas timestamp, para determinar si la fila se ha modificado después de leerla en el cursor. Si la fila se ha modificado, el intento de actualización o eliminación posicionada genera un error. No es posible especificar OPTIMISTIC si se especifica también FAST_FORWARD.

```
DECLARE cClientes CURSOR OPTIMISTIC FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

Por último, queda la opción TYPE_WARNING

- **TYPE_WARNING**
  Especifica que se envía un mensaje de advertencia al cliente si el cursor se convierte implícitamente del tipo solicitado a otro.

```
DECLARE cClientes CURSOR TYPE_WARNING FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

Podemos especificar multiples parámetros en la apertura de cursor, pero unicamente un parámetro de cada grupo. Por ejemplo:

```
DECLARE cClientes CURSOR LOCAL STATIC TYPE_WARNING FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
```

Para actualizar los datos de un cursor debemos especificar FOR UPDATE despues de la sentencia SELECT en la declaración del cursor, y WHERE CURRENT OF <nombre_cursor> en la sentencia UPDATE tal y como muestra el siguiente ejemplo.

```
-- Declaracion de variables para el cursor
DECLARE @Id int,
  @Nombre varchar(255),
  @Apellido1 varchar(255),
  @Apellido2 varchar(255),
  @NifCif varchar(20),
  @FxNacimiento datetime
-- Declaración del cursor
DECLARE cClientes CURSOR FOR
SELECT  Id, Nombre, Apellido1,
  Apellido2, NifCif, FxNacimiento
FROM CLIENTES
FOR UPDATE

-- Apertura del cursor
OPEN cClientes
-- Lectura de la primera fila del cursor
FETCH cClientes
INTO @id, @Nombre, @Apellido1, @Apellido2, @NifCif, @FxNacimiento

WHILE (@@FETCH_STATUS = 0 )
BEGIN
```

```
        UPDATE Clientes

        SET APELLIDO2 = isnull(@Apellido2,'') + ' - Modificado'

        WHERE CURRENT OF cClientes

        -- Lectura de la siguiente fila del cursor

        FETCH cClientes
        INTO @id, @Nombre, @Apellido1, @Apellido2,
            @NifCif, @FxNacimiento

    END
    -- Cierre del cursor
    CLOSE cClientes
    -- Liberar los recursos
    DEALLOCATE cClientes
```

SQL dinámico en Transact SQL

Transact SQL permite dos formas de ejecutar SQL dinamico(construir sentencias SQL dinamicamente para ejecutarlas en la base de datos):

- La instrucción **EXECUTE -** o simplemente **EXEC**
- El procedimiento almacenado **sp_executesql**

Desde aquí recomendamos la utilización de sp_**executesql** si bien vamos a mostrar la forma de trabajar con ambos métodos.

La instrucción EXECUTE

La instrucción EXECUTE - o simplemente EXEC - permite ejecutar una cadena de caracteres que representa una sentencia SQL. La cadena de caracteres debe ser de tipo **nvarchar** .
El siguiente ejemplo muestra como ejecutar una cadena de caracteres con la instrucción **EXEC**.

```
DECLARE @sql nvarchar(1000)

SET @sql = 'SELECT
            COD_PAIS,
            NOMBRE_PAIS,
            ACTIVO,
            FX_ALTA
        FROM
            PAISES'
EXEC (@sql)
```

También con SQL dinamico podemos ejecutar sentencias de tipo DDL (Data Definition Languaje), como CREATE TABLE.

```
DECLARE @sql nvarchar(1000)
SET @sql='CREATE TABLE TEMPORAL
  ( ID int IDENTITY, DATO varchar(100))'
EXEC (@sql)
```

```
SET @sql = 'SELECT * FROM TEMPORAL'

EXEC (@sql)
```

El principal incoveniente de trabajar con la instrucción **EXEC** es que no permite el uso de parametros abriendo la puerta a potenciales ataques de Sql Injections - http://www.devjoker.com/contenidos/Articulos/45/Seguridad-en-Internet--SQL-Injections.aspx

Además el uso de la instrucción **EXEC** es menos eficiente, en terminos de rendimiento, que **sp_executesql**.

Para solventar el problema debemos trabajar siempre con **sq_executesql**, que permite el uso de parametros y con el que obtendremos un mejor rendimiento de nuestras consultas.

El procedimiento almacenado sp_executesql

Para ejecutar sql dinamico, se recomienda utilizar el procedimiento almacenado **sp_executesql**, en lugar de una instrucción **EXECUTE**.

- **sp_executesql** admite la sustitución de parámetros
- **sp_executesql** es más seguro y versátil que **EXECUTE**
- **sp_executesql** genera planes de ejecución con más probabilidades de que SQL Server los vuelva a utilizar, es más eficaz que **EXECUTE**.

El siguiente ejemplo muestra el uso (muy simple) de **sp_executesql**.

```
DECLARE @sql nvarchar(1000)

SET @sql = 'SELECT
            COD_PAIS,
            NOMBRE_PAIS,
            ACTIVO,
            FX_ALTA
      FROM
            PAISES'

EXEC sp_executesql @sql
```

sp_executesql admite la sustitución de valores de parámetros para cualquier parámetro especificado en la cadena Transact-SQL a ejecutar.

El siguiente ejemplo muestra el uso de **sp_executesql** con parámetros:

```
DECLARE @sql nvarchar(1000),

  @paramDefinition nvarchar(255),

  @paramValue char(3)



SET @paramDefinition = '@codPais char(3)'

SET @paramValue = 'ESP'
```

```
SET @sql = 'SELECT
            COD_PAIS,
            NOMBRE_PAIS,
            ACTIVO,
            FX_ALTA
      FROM
            PAISES
      WHERE COD_PAIS = @codPais'

EXEC sp_executesql @sql, @paramDefinition, @paramValue
```