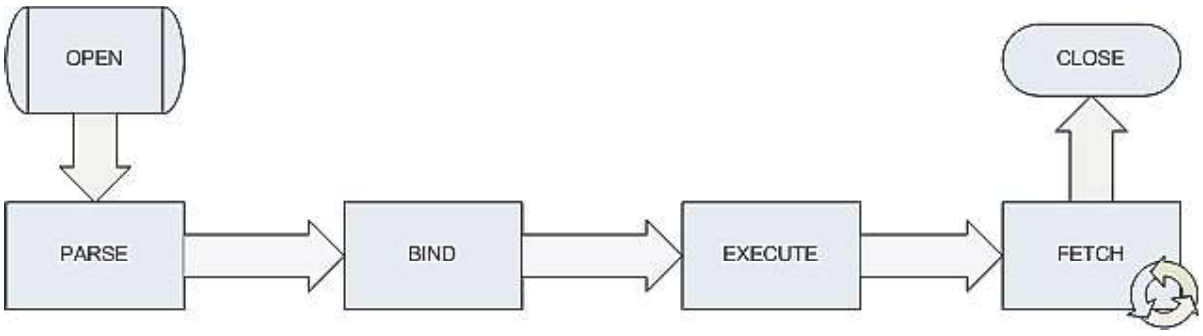
Concepto de Cursores

Una base de datos relaciona como SQL Server está orientada a conjuntos de manera natural, por ejemplo una instrucción SELECTregresa un conjunto de datos; sin embargo muchas veces es necesario utilizar no el enfoque de conjunto de datos sino de registros para realizar ciertas operaciones no complejas, es donde se implementan los cursores.

Un cursor es un objeto de la Base de Datos usado por las aplicaciones para manipular los datos fila a fila, en lugar de hacerlo en bloques de filas como lo hacen los comandos SQL normales.

Para poder trabajar con Cursores, debemos realizar los siguientes pasos:

- Declarar el Cursor con la instrucción DECLARE.
- Abrir el Cursor con la la instrucción OPEN.
- Recuperar las filas desde el Cursor con la instrucción FETCH.
- Procesar las filas obtenidas (modificar, eliminar) con la instrucción UPDATE / DELETE tabla WHERE CURRENT OF nombre_cursor.
- Cerrar el Cursor con la instrucción CLOSE.
- Eliminar la referencia al Cursor y liberar memoria con la instrucción DEALLOCATE.



Modelo de Cursores

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 siguiente 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 >
```

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 NEXTes 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 realizamos la lectura del cursor varia, debiendo utilizar la siguiente sintaxis: FETCH [NEXT | PRIOR | FIRST | LAST | RELATIVE | ABSOLUTE] FROM <>

```
--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,@FxNacimie
        nto
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 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
```

Tipos de Cursores

ODBC y ADO definen cuatro tipos de cursores admitidos por MicrosoftSQL Server. La instrucción DECLARE CURSOR se ha ampliado para que pueda especificar cuatro tipos para los cursores de Transact–SQL. Estos cursores varían en su capacidad para detectar cambios en el conjunto de resultados y en los recursos que consumen, como la memoria y el espacio de **tempdb**. Un cursor puede detectar cambios en las filas sólo cuando intenta recuperarlas una segunda vez. El origen de datos no puede notificar al cursor las modificaciones realizadas en las filas recuperadas actualmente. El nivel de aislamiento de la transacción influye también en la capacidad de un cursor para detectar los cambios.

Los cuatro tipos de cursor de servidor de la API que admite el servidor SQL Server son:

- **Cursores estáticos (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. Detectan pocos cambios o ningún cambio, y consumen relativamente pocos recursos al desplazarse.

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

- **Cursores dinámicos (DINAMIC)**

  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. Detectan todos los cambios pero consumen más recursos al desplazarse.

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

•Cursores de sólo avance (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
```

•Cursores controlados por conjunto de claves (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. Se encuentran entre los dos anteriores, pero con un consumo menor que los cursores dinámicos.

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

Aunque los modelos de cursor de la API de base de datos consideran el cursor de sólo avance como un tipo más, SQL Server no establece esta distinción. SQL Server considera que las opciones de desplazamiento de sólo avance y de desplazamiento son las que se pueden aplicar a los cursores estáticos, a los controlados por conjunto de claves y a los dinámicos.

Creación de Cursores

La sintaxis de declaración de un cursor es la siguiente:

```
declare cursor_prueba cursor for
selec nombres, apellidos from persona

/*ahora declaramos las variables con las que vamos a recorrer el cursor:*/

declare @nombres varchar(25)
declare @apellidos varchar(25)

/*Abrimos el cursor para iniciar el recorrido del mismo*/
open cursor_prueba

/*Se mueve al siguiente registro dentro del cursor y los asignamos a las variables antes
declaradas*/

fetch next from cursor_prueba
into @nombres, apellidos

/*Retorna el estatus del último registro recorrido en el cursor, cuando es igual a 0 encontró
registro pendientes de recorrer*/
while @@fetch_status = 0
begin

print 'El Nombre de la persona es: ' + @nombres + ' y sus apellidos: ' + apellidos

/*Se mueve al siguiente registro dentro del cursor*/
fetch next from cursor_prueba
into @nombres, apellidos

end

/* Cuando concluimos con el recorrido del cursor, este debe ser cerrado y luego destruído
```

```
mediante las siguientes sentencias:*/
close cursor_prueba --Cierra el cursor.
Deallocate cursor_prueba --Lo libera de la memoria y lo destruye.
```

Los cursores son muy eficientes para utilizarlos en Job de las base de datos que realizen alguna operación donde necesitemos modificar alguna información dentro de un bucle. Los cursores demandan mucho del servidor de base datos, por lo tanto, no es recomendable abusar del mismo, ya que necesitan bastante recursos para su ejecución

Utilización de Cursores

En una Base de Datos llamada A**lmacen_carnet**, añadiremos una tabla Nombrecito mediante el siguiente script:

```
create table nombrecito(
cod int,
nombre varchar(25),
apellido varchar(25),
nombrecompleto varchar(50))

insert into nombrecito values('1','Carlos','Castro','no')
insert into nombrecito values('2','Jose','Abarca','no')
insert into nombrecito values('3','Lissette','Jimenez','no')
insert into nombrecito values('4','Juan','Elias','no')
```

Como puede ver se insertaron registros y el campo de nombre completo no se ingreso correctamente, por lo que tendremos que actualizar registro por registro, para ello haremos uso de cursores.

```
-- declaramos las variables
   declare @cod as int
  declare @nombre as varchar(25)
  declare @apellido as varchar(25)

-- declaramos un cursor llamado "MICURSOR". El select debe contener sólo los
campos a utilizar.
    declare MICURSOR cursor for
  select cod, nombre,apellido from nombrecito
open MICURSOR
--Avanzamos un registro y cargamos en las variables los valores encontrados
en el primer registro
  fetch next from MICURSOR
  into @cod, @nombre, @apellido
      while @@fetch_status = 0
    begin
   update nombrecito set nombrecompleto= @nombre+' '+@apellido where cod=@cod
    -- Avanzamos otro registro
fetch next from MICURSOR
    into @cod, @nombre, @apellido
    end
   -- cerramos el cursor
        close MICURSOR
  deallocate MICURSOR
```