Taller	SQL SERVER PROGRAMACIÓN
Docente	Mg. Ing. Eric Gustavo Coronel Castillo
Tema	**CURSORES**

TRABAJANDO CON CURSORES

Declaración

Sintaxis ISO

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR
    FOR select_statement
    [ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ] [;]
```

Sintaxis Transact-SQL Extended

```
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 ] ] ] [;]
```

Ejemplo 1

```
DECLARE cur_cursos CURSOR
FOR
    SELECT cur_id, cur_nombre, cur_precio
    FROM dbo.CURSO;
```

Abrir un cursor

Sintaxis

```
OPEN { { [ GLOBAL ] cursor_name } | cursor_variable_name }
```

Ejemplo 2

```
OPEN cur_cursos;
```

Recuperar filas de un cursor

Sintaxis

```
FETCH
   [ [ NEXT | PRIOR | FIRST | LAST
         | ABSOLUTE { n | @nvar } | RELATIVE { n | @nvar } ]
   FROM ]
{ { [ GLOBAL ] cursor_name } | @cursor_variable_name }
[ INTO @variable_name [ ,...n ] ]
```

Ejemplo 3

```
DECLARE @cur_id int, @cur_nombre varchar(100), @cur_precio money

FETCH NEXT FROM cur_cursos
INTO @cur_id, @cur_nombre, @cur_precio;

SELECT @cur_id, @cur_nombre, @cur_precio
```

Cerrar un cursor

Sintaxis

```
CLOSE { { [ GLOBAL ] cursor_name } | cursor_variable_name }
```

Ejemplo 4

```
CLOSE cur_cursos;
```

Liberar recursos de un cursor

Sintaxis

```
DEALLOCATE { { [ GLOBAL ] cursor_name } | @cursor_variable_name }
```

Ejemplo 5

```
DEALLOCATE cur_cursos;
```

Ejemplo 6: Ejemplo completo

```
DECLARE cur_cursos CURSOR
FOR
   SELECT cur_id, cur_nombre, cur_precio
   FROM dbo.CURSO;

OPEN cur_cursos;

DECLARE @cur_id int, @cur_nombre varchar(100), @cur_precio money

FETCH NEXT FROM cur_cursos
INTO @cur_id, @cur_nombre, @cur_precio;

SELECT @cur_id, @cur_nombre, @cur_precio

CLOSE cur_cursos;

DEALLOCATE cur_cursos;
```

CONTROL DE UN CURSOR

Variable: @@FETCH_STATUS

Devuelve el estado de la última instrucción FETCH emitida para cualquier cursor abierto en ese momento por la conexión.

Sintaxis

```
@@FETCH_STATUS
```

Valor devuelto

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 capturada.

Ejemplo 7

```
DECLARE cur_cursos CURSOR
FOR
    SELECT cur_id, cur_nombre, cur_precio
    FROM dbo.CURSO;

OPEN cur_cursos;

DECLARE @cur_id int, @cur_nombre varchar(100), @cur_precio money

FETCH NEXT FROM cur_cursos
INTO @cur_id, @cur_nombre, @cur_precio;

WHILE( @@FETCH_STATUS = 0 )
BEGIN
    PRINT CONCAT(@cur_id, ' - ', @cur_nombre, ' - ', @cur_precio);
    FETCH NEXT FROM cur_cursos
    INTO @cur_id, @cur_nombre, @cur_precio;
END;

CLOSE cur_cursos;
```

```
DEALLOCATE cur_cursos;
```

Variable: @@CURSOR_ROWS

Devuelve el número de filas certificadas que se encuentran en el último cursor abierto en la conexión actual. Para mejorar el rendimiento, SQL Server puede rellenar asincrónicamente los cursores estáticos y de conjunto de claves de gran tamaño. Puede llamar a @@CURSOR_ROWS para determinar que el número de filas que cumplan las condiciones del cursor se recuperen en el momento en que se llama a @@CURSOR_ROWS.

Sintaxis

```
@@CURSOR_ROWS
```

Valor devuelto

Valor devuelto	Descripción
-m	El cursor se rellena de forma asincrónica. El valor devuelto (-m) es el número de filas que el conjunto de claves contiene actualmente.
-1	El cursor es dinámico. Como los cursores dinámicos reflejan todos los cambios, el número de filas correspondientes al cursor cambia constantemente. Nunca se puede afirmar que se han recuperado todas las filas que correspondan.
0	No se han abierto cursores, no hay filas calificadas para el último cursor abierto, o éste se ha cerrado o su asignación se ha cancelado.
n	El cursor está completamente relleno. El valor devuelto (n) es el número total de filas del cursor.

Ejemplo 8

El este ejemplo se declara un cursor y se utiliza la sentencia PRINT para mostrar el valor de @@CURSOR_ROWS.

@@CURSOR_ROWS, tiene el valor 0 antes de abrir el cursor y el valor -1 después de abrir el cursor, lo que indica que el número de filas es dinámico, cambia constantemente.

```
DECLARE cur_cursos CURSOR
FOR
    SELECT cur_id, cur_nombre, cur_precio
    FROM dbo.CURSO;
```

```
PRINT CONCAT('CURSOR_ROWS = ', @@CURSOR_ROWS);

OPEN cur_cursos;

PRINT CONCAT('CURSOR_ROWS = ', @@CURSOR_ROWS);

CLOSE cur_cursos;

DEALLOCATE cur_cursos;
GO

CURSOR_ROWS = 0
CURSOR_ROWS = -1
```

Función: CURSOR_STATUS ( )

Una función escalar que permite a quien llama a un procedimiento almacenado determinar si el procedimiento ha devuelto un cursor y el conjunto de resultados de un parámetro determinado.

Sintaxis

```
CURSOR_STATUS
   (
         { 'local' , 'cursor_name' }
         { 'global' , 'cursor_name' }
         | { 'variable' , 'cursor_variable' }
   )
```

Valor devuelto

Valor devuelto	Nombre de cursor	Variable de cursor
1	El conjunto de resultados del cursor tiene al menos una fila. Para los cursores INSENSITIVE y de conjunto de claves, el conjunto de resultados tiene al menos una fila. Para los cursores dinámicos, el conjunto de resultados puede tener cero, una o más filas.	El cursor asignado a esta variable está abierto. Para los cursores INSENSITIVE y de conjunto de claves, el conjunto de resultados tiene al menos una fila. Para los cursores dinámicos, el conjunto de resultados puede tener cero, una o más filas.

0	El conjunto de resultados del cursor está vacío.*	El cursor asignado a esta variable está abierto, pero el conjunto de resultados está definitivamente vacío.*
-1	El cursor está cerrado.	El cursor asignado a esta variable está cerrado.
-2	No aplicable.	Puede ser: El procedimiento llamado anteriormente no ha asignado ningún cursor a esta variable OUTPUT. El procedimiento llamado anteriormente asignó un cursor a esta variable OUTPUT, pero se encontraba en un estado cerrado al terminar el procedimiento. Por tanto, se cancela la asignación del cursor y no se devuelve al procedimiento que hace la llamada. No hay ningún cursor asignado a una variable declarada de cursor.
-3	No existe ningún cursor con el nombre indicado.	No existe una variable de cursor con el nombre indicado o, si existe, no tiene todavía ningún cursor asignado.

Ejemplo 9

El siguiente script muestra el estado del cursor antes y después de la apertura.

```
-- Seleccionando la base de datos

USE EduTec;
GO

-- Crea un cursor

DECLARE cur_demo CURSOR
FOR SELECT * FROM dbo.Curso;

-- Cursor cerrado

SELECT CURSOR_STATUS('global','cur_demo') AS 'Después de declarar';

-- Cursor abierto

OPEN cur_demo;
SELECT CURSOR_STATUS('global','cur_demo') AS 'Después de abrir';

-- Cursor cerrado
```

```
CLOSE cur_demo;
SELECT CURSOR_STATUS('global','cur_demo') AS 'Después de cerrar';

-- Remover el cursor

DEALLOCATE cur_demo;
GO
```

Se obtiene el siguiente resultado:

```
Después de declarar
-------------------
-1

(1 filas afectadas)

Después de abrir
----------------
1

(1 filas afectadas)

Después de cerrar
-----------------
-1

(1 filas afectadas)
```

BUCLE DE EXTRACCIÓN

Plantilla

```
FETCH NEXT FROM <nombre_cursor> INTO <lista_variables>;
WHILE ( @@FETCH_STATUS = 0 )
BEGIN

    --   Proceso

    FETCH NEXT FROM <nombre_cursor> INTO <lista_variables>;
END
```

Ejemplo 10

El siguiente script muestra la cantidad de empleados que hay en cada departamento:

```
USE RH;
GO

DECLARE cur_depts CURSOR
FOR
    SELECT
           d.iddepartamento codido,
           d.nombre nombre,
           count(*) emps
    FROM dbo.departamento d
    join dbo.empleado e
    on d.iddepartamento = e.iddepartamento
    group by d.iddepartamento, d.nombre;

OPEN cur_depts;

DECLARE @codigo int, @nombre varchar(100), @emps int

PRINT CONCAT('COD', SPACE(4),
          LEFT('NOMBRE' + SPACE(20), 20), SPACE(4), 'EMPS');
PRINT '-----------------------------------';

FETCH NEXT FROM cur_depts INTO @codigo, @nombre, @emps;
WHILE( @@FETCH_STATUS = 0 )
BEGIN
```

```
    PRINT CONCAT(@codigo, SPACE(4),
          LEFT(@nombre + SPACE(20), 20), SPACE(4), @emps);
    FETCH NEXT FROM cur_depts INTO @codigo, @nombre, @emps;
END;

CLOSE cur_depts;
DEALLOCATE cur_depts;
GO
```

El resultado que se obtiene es similar al siguiente:

```
COD    NOMBRE                  EMPS
-----------------------------------
100    Gerencia                2
101    Contabilidad            3
102    Investigacion           6
103    Ventas                  7
105    Sistemas                4
```

EJERCICIOS

Ejercicio 1

En la base de datos RH, crear un procedimiento que de cada departamento muestre el trabajador con menor salario y el trabajador con mayor tiempo de servicio. Se deben mostrar los empates.

Ejercicio 2

En la base de EDUCA, crear un procedimiento que muestre el alumno con mayor nota. Se deben mostrar los empates.

USO DE TABLAS TEMPORALES

Variables de tipo tabla

Las Variables de Tabla que son un tipo de datos que puede ser utilizados en un lote Transact-SQL (Batch), procedimiento almacenado o función; estas variables de tabla son creadas y definidas de forma similar a una tabla, sólo que tienen un alcance de vida bien definido. Las Variables de tabla suelen ser buenos reemplazos de tablas temporales siempre y cuando el conjunto de datos es pequeño.

Razones para usar las variables de tabla:

- **Duración o alcance.** La duración de la variable de tabla sólo vive durante la ejecución del lote, función, o procedimiento almacenado.
- **Tiempos de bloqueo más cortos.** Por el estrecho alcance o tiempo de vida.
- **Menos re compilaciones.** Cuando se usa en los procedimientos almacenados.

El inconveniente de utilizar las variables de tabla es su rendimiento. El rendimiento de las variables de tabla se ve afectado cuando el resultado es demasiado grande o cuando los datos de la columna de cardinalidad son fundamentales para la optimización del proceso de consulta.

La sintaxis para crear una variable de tabla es similar a la de crear una tabla normal, se utiliza la palabra clave DECLARE y el nombre de tabla, anteponiendo el símbolo @.

Sintaxis:

```
DECLARE @TableName TABLE
(
    column_name <data_type> [ NULL | NOT NULL ]
    [ ,...n ]
)
```

Ejemplo 11

En el siguiente script se crea una variable tipo TABLE, se inserta valores y finalmente se consulta.

```
-- Creando la variable de tipo tabla.
DECLARE @Catalogo TABLE
(
    idProd int NOT NULL PRIMARY KEY,
    nombre varchar(30) NULL,
```

```
   precio money
);

-- Insertando datos en la variable de tipo tabla.
INSERT INTO @Catalogo VALUES
(1, 'Refrigeradora', 2400.0),
(2, 'Televisor', 3500.0),
(3, 'Laptop', 4500.0);

-- Consultando datos de la variable de tipo tabla.
SELECT * FROM @Catalogo;
GO
```

El resultado que se obtiene es el siguiente:

```
idProd      nombre                         precio
----------- ------------------------------ ---------------------
1           Refrigeradora                  2400.00
2           Televisor                      3500.00
3           Laptop                         4500.00

(3 filas afectadas)
```

Al terminar la ejecución del batch o bloque de instrucciones automáticamente se eliminara la variable tabla.

Tablas temporales locales

Las tablas temporales locales están disponibles para usarse en la sesión actual. Varias conexiones pueden crear una tabla temporal con mismo nombre, esto solo para tablas temporales locales, sin causar conflictos. La representación interna de la tabla local tiene un nombre único, para no estar en conflicto con otras tablas temporales con el mismo nombre creado por otras conexiones en la base de datos tempdb.

Las tablas temporales locales son eliminadas con el comando DROP o se eliminan automáticamente de memoria cuando se cierra la conexión del usuario.

Las tablas temporales locales se crean anteponiendo el símbolo # al nombre de la tabla.

Ejemplo 12

Creación de la tabla temporal.

```
-- Creación de la tabla temporal
```

```
CREATE TABLE #ResumenVentas
(
   idProd int NOT NULL PRIMARY KEY,
   nombre varchar(30) NULL,
   ventas money NULL
);
GO
```

Insertando datos en la tabla temporal, desde la misma sesión.

```
-- Insertando datos
INSERT INTO #ResumenVentas VALUES
(1, 'Refrigeradora', 45657.0),
(2, 'Televisor', 65350.0),
(3, 'Laptop', 145640.0);
GO
```

Consultando la tabla temporal.

```
-- Consultando la tabla temporal
SELECT *  FROM #ResumenVentas;
```

Resultado obtenido.

```
idProd      nombre                         ventas
----------- ------------------------------ --------------------
1           Refrigeradora                  45657.00
2           Televisor                      65350.00
3           Laptop                         145640.00

(3 filas afectadas)
```

Eliminado la tabla temporal.

```
-- Eliminando la tabla
DROP TABLE #ResumenVentas;
GO
```

Para que el ejemplo funcione las instrucciones deben ejecutarse en la misma sesión.

Tablas temporales globales

Las tablas temporales globales tienen un alcance diferente al de las tablas temporales locales. Una vez que se crea una tabla temporal global en una sesión, cualquier usuario con permisos adecuados sobre la base de datos puede acceder a la tabla desde cualquier otra sesión. A diferencia de tablas temporales locales, no se pueden crear versiones simultáneas de una tabla temporal global, ya que esto generará un conflicto de nombres.

Las tablas temporales globales de eliminan explícitamente de SQL Server ejecutando DROP TABLE. También se eliminan automáticamente después de que se cierra la sesión que la crea, la tabla temporal global no es referenciada por otras conexiones, pero es muy raro ver que se utilicen tablas temporales globales en bases de datos en producción.

Es importante considerar cuando una tabla va o debe ser compartida a través de conexiones, se debe crear una tabla real, en lugar de una tabla temporal global. No obstante, SQL Server ofrece esto como una opción.

Las tablas temporales locales se crean anteponiendo el símbolo ## al nombre de la tabla.

Ejemplo 13

Creando la tabla temporal global.

```
--Creando la tabla temporal global
CREATE TABLE ##Autores
( id int NOT NULL IDENTITY(1,1) PRIMARY KEY,
  nombre varchar(30) NULL
);
GO
```

Insertar datos en la tabla temporal.

```
-- Insertando datos
INSERT INTO ##Autores(nombre) VALUES
('Gustavo Coronel'),
('Sergio Matsukawa'),
('Ricardo Marcelo');
GO
```

La consulta a esta tabla se puede hacer desde cualquier otra sesión.

```
-- Consultando la tabla temporal
select * from ##Autores;
GO
```

El resultado es el siguiente.

```
id          nombre
----------- ------------------------------
1           Gustavo Coronel
2           Sergio Matsukawa
3           Ricardo Marcelo

(3 filas afectadas)
```

EJERCICIOS

Ejercicio 3

En la base de datos RH, crear un procedimiento que de cada departamento muestre lo siguiente:

- La cantidad de empleados
- Planilla sin comisión
- Planilla con comisión

Ejercicio 4

En la base de datos EDUTEC, crear un procedimiento que reciba como parámetro un periodo, por ejemplo 2010, 2011, 2012, etc. y reporte por cada ciclo en el periodo los siguientes datos:

- Cantidad de cursos programados
- Cantidad de alumnos proyectados
- Cantidad de alumnos matriculados
- Importe proyectado (de alumnos proyectados)
- Importe real (de alumnos matriculados)