

## ***Capítulo 10***

### ***Procedimientos almacenados y cursores***

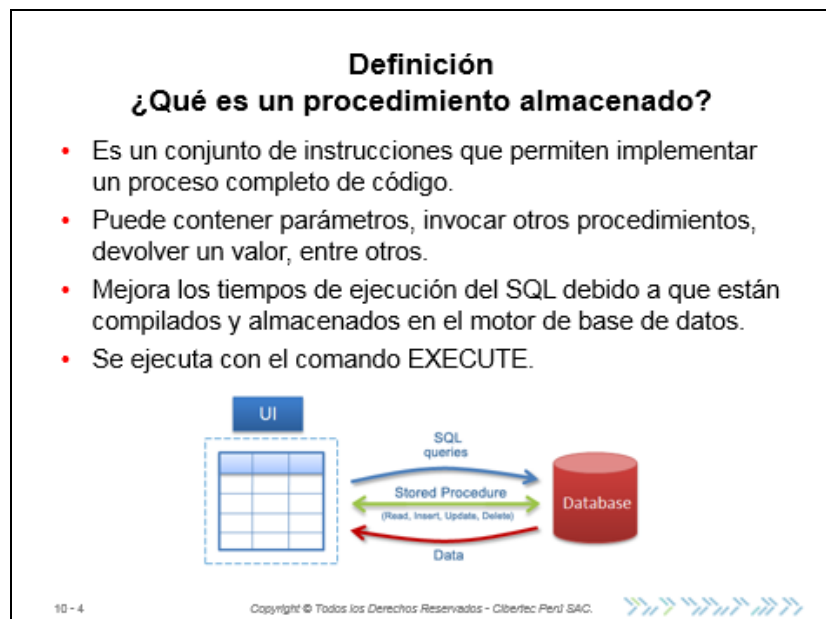
**Al finalizar el capítulo, el alumno podrá:**

- Construir procedimientos almacenados con T-SQL para la consulta y actualización de datos.

**Temas:**

1. Definición
2. Procedimientos almacenados del usuario
3. Tipos de parámetros
4. Procedimientos almacenados anidados
5. Cursor

## 1. Definición



### 1.1. Definición

Los procedimientos almacenados son grupos formados por instrucciones SQL y el lenguaje de control de flujo. Cuando se ejecuta un procedimiento, se prepara un plan de ejecución para que la subsiguiente ejecución sea muy rápida.

Los procedimientos almacenados pueden:

- Incluir parámetros.
- Llamar a otros procedimientos.
- Devolver un valor de estado a un procedimiento de llamada o lote para indicar el éxito o fracaso de este, y la razón de dicho fallo.
- Devolver valores de parámetros a un procedimiento de llamada o lote.
- Ejecutarse en SQL Server remotos.

La posibilidad de escribir procedimientos almacenados mejora notablemente la potencia, eficacia y flexibilidad de SQL. Los procedimientos compilados mejoran la ejecución de las instrucciones y lotes de SQL de forma dramática. Además, los procedimientos almacenados pueden ejecutarse en otro SQL Server, si el servidor del usuario y el remoto están configurados para permitir logins remotos.

Los procedimientos almacenados se diferencian de las instrucciones SQL ordinarias y de los lotes de instrucciones SQL en que están precompilados. La primera vez que se ejecuta un procedimiento, el procesador de consultas de SQL Server, lo analiza y prepara un plan de ejecución que se almacena de forma definitiva en una tabla del sistema. Posteriormente, el procedimiento se ejecuta según el plan almacenado. Puesto que ya se ha realizado la mayor parte del trabajo de procesamiento de consultas, los procedimientos almacenados se ejecutan casi de forma instantánea.

Algunas reglas adicionales para la creación de procedimientos almacenados son:

- Las instrucciones `create procedure` no pueden combinarse con otras instrucciones en un solo lote.
- La definición `create procedure` propiamente dicha, puede incluir cualquier número y tipo de instrucción SQL, con la excepción de **use** y las siguientes instrucciones **create**:
  - `CREATE VIEW`
  - `CREATE DEFAULT`
  - `CREATE RULE`
  - `CREATE TRIGGER`
  - `CREATE PROCEDURE`
- Se pueden crear otros objetos de base de datos dentro de un procedimiento. Es posible hacer referencia a un objeto creado en el mismo procedimiento, siempre que se cree antes de hacer referencia a él. La instrucción `CREATE` del objeto debe ocupar la primera posición en el orden real de las instrucciones del procedimiento.
- En un procedimiento almacenado, no es posible crear un objeto, omitirlo y después, crear otro con el mismo nombre.
- SQL Server crea los objetos definidos en un procedimiento almacenado cuando el procedimiento se ejecuta, no cuando se compila.
- Si se ejecuta un procedimiento que llama a otro, el procedimiento llamado puede acceder a los objetos creados por el primer procedimiento.
- Si se crea una tabla temporal dentro de un procedimiento, la tabla solo existirá para ese procedimiento y desaparecerá al salir de este.
- El número máximo de parámetros de un procedimiento almacenado es de 255.
- El número máximo de variables locales y globales de un procedimiento solo está limitado por la memoria disponible.

## 1.2. Ventajas

- **Reutilización de código**, el encapsulamiento de un procedimiento almacenado es óptimo para reutilizar su código. Se elimina la necesidad de escribir el mismo código, se reducen inconsistencias en el código y permite que cualquier usuario ejecute el código sin tener acceso a los objetos que hace referencia.
- **Mayor seguridad**, se pueden ejecutar procedimientos almacenados con instrucciones que hacen referencia a objetos que los usuarios no tienen permisos. El procedimiento almacenado realiza la ejecución del código y todas las instrucciones y controla el acceso a los objetos a los que hace referencia. Esto hace más sencillo la asignación de permisos.
- **Tráfico de red reducido**, un procedimiento almacenado se ejecuta en un único lote de código. Esto reduce el tráfico de red cliente servidor porque únicamente se envía a través de la red, la llamada que ejecuta el procedimiento almacenado. La encapsulación del código del procedimiento almacenado permite que viaje a través de la red como un solo bloque.
- **Mantenimiento mas sencillo**, se puede trabajar en los aplicativos en base a capas, cualquier cambio en la base de datos, hace sencillo los cambios en los procedimientos almacenados que hacen uso de los objetos cambiados en la base de datos.
- **Rendimiento mejorado**, los procedimientos almacenados se compila la primera vez que se ejecutan y crean un plan de ejecución que vuelve a usar en posteriores ejecuciones.

### 1.3. Tipos de procedimiento

- **Definidos por el usuario**, se crea por el usuario en las bases de datos definidas por el usuario o en las de sistema (master, tempdb, model, msdb).
- **Del sistema**, son propios del SQL Server. Los caracteres iniciales de estos procedimientos son "sp\_" la cual no se recomienda para los procedimientos almacenados definidos por el usuario.
- **Temporales**, son procedimientos definidos por el usuario, estos se almacenan en la base de datos tempdb. Existen dos tipos de procedimientos almacenados temporales: locales (primer carácter es #) y globales (primer carácter ##). Se diferencian entre sí por los nombres, la visibilidad y la disponibilidad. Los procedimientos temporales locales tienen como primer carácter de sus nombres un solo signo de número (#); solo son visibles en la conexión actual del usuario y se eliminan cuando se cierra la conexión. Los procedimientos temporales globales presentan dos signos de número (##) antes del nombre; lo pueden usar todos los usuarios conectados, se eliminan cuando se desconectan todos los usuarios.
- **Extendidos definidos por el usuario**, los procedimientos extendidos tienen instrucciones externas en un lenguaje de programación como puede ser C. Estos procedimientos almacenados son DLL que una instancia de SQL Server puede cargar y ejecutar dinámicamente.



**Aclaración del autor:**

SQL Server proporciona una gran variedad de procedimientos almacenados como herramientas adecuadas para el usuario. Estos procedimientos almacenados se llaman **procedimientos del sistema** y escapan al alcance del presente manual.

## 2. Procedimientos almacenados del usuario

### Procedimientos almacenados del usuario

#### ¿Qué es un parámetro o argumento?

- Es un argumento que el usuario debe suministrar un valor al ejecutar el procedimiento.
- Los nombres deben empezar con "@", luego el tipo de dato.
- Permiten enviar valores dando flexibilidad en la ejecución.
- Se puede asignar un valor predeterminado, si el usuario no lo indica al invocar al procedimiento almacenado.

10 - 7

Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.



Los procedimientos almacenados se crean con la instrucción **CREATE PROCEDURE**.

Para ejecutar un procedimiento almacenado, ya sea un procedimiento del sistema o uno definido por el usuario, se debe usar el comando EXECUTE. También puede utilizar el nombre del procedimiento almacenado, siempre que sea la primera palabra de una instrucción o lote.

La sintaxis para la creación o modificación de un procedimiento almacenado sencillo, sin funciones especiales como parámetros, es como se muestra.

```
CREATE PROCEDURE NOMBRE_PROCEDIMIENTO  
AS SQL_statements
```

Los procedimientos almacenados son objetos de base de datos y sus nombres deben ajustarse a las reglas para identificadores. Es posible incluir cualquier número y cualquier tipo de instrucción SQL, salvo las instrucciones CREATE. Un procedimiento puede ser tan sencillo como una sola instrucción que enumere los nombres de todos los usuarios de una base de datos.

```
CREATE PROCEDURE usp_ListaProducto  
AS  
    SELECT ProductID, ProductName, QuantityPerUnit, UnitPrice, UnitsInStock  
    FROM Products
```

Para ejecutar un procedimiento almacenado, se debe emplear la palabra clave **EXECUTE** o **EXEC** y el nombre del procedimiento almacenado o simplemente, especificar el nombre del procedimiento, siempre que se envíe a SQL Server o sea la primera instrucción de un lote. Además, se puede ejecutar LISTACIENTES de cualquiera de las siguientes formas.

```
EXECUTE usp_ListaProducto  
EXEC usp_ListaProducto
```

Para ejecutar un procedimiento almacenado en un SQL Server remoto, debe proporcionar el nombre del servidor. La sintaxis completa de una llamada de procedimiento remoto es como se muestra a continuación.

```
EXEC NOMB_SERVID.[BASE_DATOS].[PROPIETARIO].PROCEDURE_NAME
```

Los siguientes ejemplos ejecutan el procedimiento usp\_ListaProducto en la base de datos Northwind en el servidor CIBERTEC.

```
EXECUTE CIBERTEC.Northwind.usp_ListaProducto  
EXEC CIBERTEC.Northwind.usp_ListaProducto
```

El nombre de la base de datos es opcional solo si el procedimiento almacenado se encuentra en la base de datos predeterminada del usuario. Asimismo, el nombre del propietario es opcional solo si el propietario de la base de datos ("DBO") es el propietario del procedimiento o si el usuario lo posee. Lógicamente, es necesario tener permiso para ejecutar el procedimiento.

Si se desea modificar un procedimiento almacenado, se debe recurrir al comando **ALTER PROCEDURE**. Por ejemplo, al procedimiento almacenado usp\_ListaProducto se desea agregar el campo DaysToManufacture, se realiza lo siguiente.

```
ALTER PROCEDURE usp_ListaProducto  
AS  
    SELECT ProductID, ProductName, QuantityPerUnit, UnitPrice,  
           UnitsInStock, Discontinued  
    FROM Products
```

Si se desea eliminar el procedimiento emplee el comando **DROP PROCEDURE**, seguido del nombre del procedimiento a eliminar. Por ejemplo:

```
DROP PROCEDURE usp_ListaProducto
```

Existe una nueva sentencia T-SQL **CREATE OR ALTER** para crear o modificar un procedimiento almacenado, sin necesidad de saber si este objeto de la base de datos existe o no, donde trabajará como una sentencia **CREATE** normal si el objeto no existe o trabajará como una sentencia **ALTER** normal si el objeto ya existe.

```
CREATE OR ALTER PROCEDURE NOMBRE_PROCEDIMIENTO  
AS SQL_statements
```

Un procedimiento puede incluir más de una instrucción.

```
CREATE PROCEDURE usp_showall
AS
    SELECT COUNT(*) FROM SYSUSERS
    SELECT COUNT(*) FROM SYSOBJECTS
    SELECT COUNT(*) FROM SYSCOLUMNS
```

Cuando se ejecuta el procedimiento, los resultados de cada comando se muestran en el orden en que la instrucción aparece en el procedimiento.

```
-----
14
(1 fila afectada)

-----
182
(1 fila afectada)

-----
1199
(1 fila afectada)
```

Cuando el comando **CREATE PROCEDURE** se ejecuta de forma correcta, el nombre del procedimiento se almacena en **SYSOBJECTS** y su texto en **SYSCOMMENTS**.

Se puede mostrar el texto de un procedimiento con el procedimiento del sistema `sp_helptext`.

```
sp_helptext usp_showall
```

```
Text
-----
CREATE    PROCEDURE usp_showall
AS
    SELECT COUNT(*) FROM SYSUSERS
    SELECT COUNT(*) FROM SYSOBJECTS
    SELECT COUNT(*) FROM SYSCOLUMNS
```



**Aclaración del autor:**

Los procedimientos almacenados pueden servir de mecanismos de seguridad, ya que un usuario puede recibir el permiso para ejecutar un procedimiento almacenado, aunque no tenga permisos en las tablas o vistas referenciadas en el mismo ni permiso para ejecutar comandos específicos.

### 3. Tipos de parámetros

#### Tipos de parámetros

#### Sintaxis para parámetro predeterminado

```
CREATE PROCEDURE usp_Lista_Empleados_Departamentos
  @Department varchar(50) = 'Production'
AS
SELECT FirstName, LastName, JobTitle, Department
FROM HumanResources.vEmployeeDepartment
WHERE Department = @Department
```

```
CREATE PROCEDURE usp_Lista_Empleados
  @LastName varchar(50) = 'D%',
  @FirstName varchar(50) = '%'
AS
SELECT FirstName, LastName, JobTitle, Department
FROM HumanResources.vEmployeeDepartment
WHERE FirstName LIKE @FirstName AND LastName LIKE @LastName
```

10 - 9
Copyright © Todos los Derechos Reservados - Cibertec Perú SAC.

Un parámetro es un argumento de un procedimiento almacenado. Es posible declarar uno o más parámetros de forma opcional en una instrucción **CREATE OR ALTER PROCEDURE**. El usuario debe suministrar el valor de cada parámetro indicado en una instrucción **CREATE OR ALTER PROCEDURE** al ejecutarse el procedimiento.

Los nombres de los parámetros deben estar precedidos del símbolo "@" y ajustarse a las reglas para identificadores. Es necesario asignarles un tipo de datos del sistema o uno definido por el usuario y una longitud si es necesario, para el tipo de datos. Los nombres de los parámetros son locales para el procedimiento que los crea; los mismos nombres de parámetros pueden utilizarse en otros procedimientos. Los nombres de parámetro, incluido el símbolo "@", pueden tener una longitud máxima de 30 bytes.

#### 3.1. Parámetros de entrada

Son aquellos que permiten pasar valores a los procedimientos, de tal forma que sean flexibles en su ejecución.

A continuación, se muestra un procedimiento almacenado que permite emitir la orden de un cliente, pasando como parámetro su código, de la BD Northwind.

```
CREATE PROCEDURE usp_Order_Customer
  @CustomerID char(5)
AS
  SELECT O.OrderID, O.OrderDate, O.RequiredDate, O.ShippedDate,
         SUM(OD.Quantity * OD.UnitPrice) as n_Total
  FROM Orders O
  INNER JOIN [Order Details] OD ON O.OrderID = OD.OrderID
  WHERE O.CustomerID = @CustomerID
  GROUP BY O.OrderID, O.OrderDate, O.RequiredDate, O.ShippedDate

/* EJECUCIÓN */
EXEC usp_Order_Customer 'VINET'
```



Este otro ejemplo permite saber la fecha de cumpleaños y el teléfono de un determinado empleado enviando como parámetro su código.

```
CREATE OR ALTER Procedure usp_Employee
@EmployeeID    int
As
    SELECT EmployeeID,
           FirstName,
           LastName,
           BirthDate,
           HomePhone
    FROM Employees
    WHERE EmployeeID = @EmployeeID

/* EJECUCIÓN */
EXEC usp_Employee 1
```

### 3.2 Parámetros predeterminados

Se puede asignar un valor predeterminado al parámetro de la instrucción CREATE PROCEDURE. Este valor, que puede ser cualquier constante, se toma como el argumento del procedimiento si el usuario no proporciona ninguno.

Se muestra un procedimiento en donde lista los clientes pasando como parámetro el país que pertenece. Si no se proporciona ningún país, el procedimiento asume como país 'USA'.

```
CREATE OR ALTER PROCEDURE usp_Lista_Clientes
@Country    varchar(15) = 'USA'
AS
    SELECT CustomerID, CompanyName, ContactName, Country
    FROM Customers
    WHERE Country = @Country
```

Si se espera un parámetro, pero no se suministra ninguno y no se proporciona ningún valor en la instrucción CREATE PROCEDURE, SQL Server mostrará un mensaje de error con los parámetros que espera el procedimiento.

Además, existen caracteres comodines en un parámetro predeterminado. El valor predeterminado puede incluir los caracteres comodín (% , \_ , [ ] y [ ^ ] ) si el procedimiento utiliza el parámetro con la palabra clave LIKE. Por ejemplo:

```
CREATE OR ALTER PROCEDURE usp_Lista_Empleados
@LastName    varchar(50) = 'D%',
@FirstName    varchar(50) = '%'
AS
    SELECT FirstName, LastName, Title, BirthDate, Country
    FROM Employees
    WHERE FirstName LIKE @FirstName AND LastName LIKE @LastName
```

### 3.3 Uso de más de un parámetro

El siguiente procedimiento almacenado maneja varios parámetros de entrada, para la inserción de una categoría.

```
CREATE OR ALTER PROCEDURE usp_InsertCategory
@CategoryID INT,
@CategoryName varchar(15),
@Description varchar(max)
AS
BEGIN
    INSERT INTO Categories (CategoryID, CategoryName, Description)
    VALUES (@CategoryID, @CategoryName, @Description)
END

/* EJECUCIÓN */
EXEC usp_InsertCategory 9, 'Beef', 'Prepared meats'
```

### 3.4 Parámetros de salida

Cuando las instrucciones **CREATE OR ALTER PROCEDURE** y **EXECUTE** incluyen la opción **OUTPUT** con un nombre de parámetro, el procedimiento devuelve un valor al solicitante, que puede ser un lote SQL u otro procedimiento almacenado.

El valor devuelto puede utilizarse en instrucciones adicionales en el lote o el procedimiento de llamada. Cuando usan parámetros de retorno en una instrucción **EXECUTE** que forma parte de un lote, los valores de retorno se imprimen con un encabezado antes de que se ejecuten las instrucciones subsiguientes del lote.

Este procedimiento almacenado realiza la multiplicación con dos valores enteros. El tercer valor entero, @result, se define como un parámetro output.

```
CREATE OR ALTER PROCEDURE usp_MATHTUTOR
@MULT1 INT,
@MULT2 INT,
@RESULT INT OUTPUT
AS
    SELECT @RESULT = @MULT1 * @MULT2
```

Para utilizar MATHTUTOR a fin de poner en cifras un problema de multiplicación, se debe declarar la variable @result e incluirla en la instrucción EXECUTE. La adición de la palabra clave **OUTPUT**, a la instrucción EXECUTE muestra el valor de los parámetros de retorno.

```
DECLARE @RESULT INT
EXEC usp_MATHTUTOR 5, 6, @RESULT OUTPUT
```

Asimismo, si se quisiera adivinar la respuesta y ejecutar este procedimiento proporcionando tres valores enteros, no se verían los resultados de la multiplicación. La instrucción SELECT del procedimiento asigna valores, pero no imprime.

La palabra clave **OUTPUT** puede abreviarse como **OUT**, del mismo modo que **EXECUTE** puede acortarse como **EXEC**.

Un procedimiento almacenado puede devolver varios valores; cada uno debe definirse como una variable **OUTPUT** en el procedimiento almacenado y en las instrucciones de llamada:

```
EXEC MYPROC @A = @MYVARA OUT, @B = @MYVARB OUT
```

Si se especifica **OUTPUT** mientras se ejecuta un procedimiento y el parámetro no se define utilizando **OUTPUT** en el procedimiento almacenado, aparecerá un mensaje de error.

No es un error llamar a un procedimiento que incluya especificaciones de valor de retorno, sin solicitar los valores de retorno con **OUTPUT**. Sin embargo, no se obtendrán los valores de retorno.

El autor del procedimiento almacenado controla la información a la que pueden acceder los usuarios y estos tienen control sobre sus variables.

En el siguiente ejemplo se muestra un procedimiento con un parámetro de entrada y otro de salida. El parámetro @CustomerID recibirá un valor de entrada especificado por el programa de llamada. La instrucción SELECT usa el último valor del parámetro de entrada para obtener el valor total de las órdenes (UnitPrice \* Quantity). La instrucción SELECT, también, asigna el valor al parámetro de salida @Total, que devuelve el valor al programa de llamada cuando finaliza el procedimiento.

```
CREATE OR ALTER PROCEDURE usp_GetCustomerTotal
@CustomerID      char(05),
@Total            money    OUTPUT
AS
    SELECT @Total = SUM(OD. UnitPrice * OD.Quantity)
    FROM Orders O
    JOIN [Order Details] OD ON O.OrderID = OD.OrderID
    WHERE O.CustomerID = @CustomerID

/* EJECUCIÓN */
DECLARE @Total_Output      money
EXECUTE usp_GetCustomerTotal 'VINET', @Total_Output OUTPUT
PRINT 'Year-to-date orders for this customers is ' +
    convert(varchar(10), @Total_Output)
```




## 5. Cursor

### Cursor

#### ¿Qué es un cursor de datos?

- Nombre simbólico asociado a una instrucciones T-SQL.
- Puntero a una serie de registros en memoria.
- Permite moverse a través de los resultados de una consulta.
- Una vez cargado puede accederse a las filas registro por registro.

10 - 12Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

### 5.1 Descripción de cursores

Las instrucciones de Microsoft SQL Server producen un conjunto completo de resultados, pero hay ocasiones en que los resultados se procesan mejor de fila en fila. Abrir un **cursor** sobre un conjunto de resultados permite procesar el conjunto de resultados de fila en fila.

Un **cursor** es el nombre simbólico asociado a una instrucción **SELECT** Transact-SQL mediante una instrucción de declaración. Se compone de las siguientes partes:

- **Conjunto de resultados del cursor:** el conjunto (tabla) de filas que resulta de ejecutar una consulta asociada al cursor.
- **Posición del cursor:** un puntero en una fila dentro del conjunto de resultados del cursor.

La posición del cursor indica la fila actual del cursor. Puede modificar o eliminar la fila de forma explícita utilizando las instrucciones **DELETE** o **UPDATE** con una cláusula que especifique el cursor. Puede cambiar la posición actual del cursor mediante una operación llamada **recuperación**. Una recuperación desplaza hacia abajo la posición actual del cursor una o más filas en el conjunto de resultados del cursor.

Un cursor se comporta en gran medida como un puntero de archivo hacia una serie de registros de archivos, donde el cursor actúa como un puntero hacia los resultados de la consulta. Sin embargo, los cursores solo admiten el movimiento hacia delante (o secuencial) a través de los resultados de la consulta. Una vez que se han recobrado varias filas, no es posible volver hacia atrás en el conjunto de resultados del cursor para acceder a ellas de nuevo. Este proceso permite examinar los resultados de una consulta fila por fila.

## 5.2 Estados de los cursores

Después de declarar el cursor, este se encuentra en uno de estos dos estados:

- **Cerrado:** el conjunto de resultados del cursor no existe, por lo que no es posible leer información en él. Los cursores se encuentran inicialmente en este estado. Es necesario abrir el cursor de forma explícita para poder utilizarlo. Una vez abierto, se puede cerrar de forma explícita después de terminar. SQL Server puede cerrar un cursor de forma implícita por varias razones, explicadas posteriormente en este capítulo.
- **Abierto:** las filas del conjunto de resultados del cursor se encuentran disponibles para su lectura o actualización.

Se puede cerrar un cursor y después volver a abrirse. La reapertura de un cursor vuelve a crear el conjunto de resultados del cursor y coloca el cursor delante de la primera fila. Esto permite progresar por un conjunto de resultados del cursor, tantas veces como sea necesario. El cursor se puede cerrar en cualquier momento, sin necesidad de examinar todo su conjunto de resultados.

Todas las operaciones de cursor, como recobrar o actualizar una fila, se llevan a cabo en referencia a la posición actual del cursor. Actualizar una fila del cursor implica modificar los datos de la fila o eliminar esta por completo. No es posible utilizar los cursores para insertar filas. Todas las actualizaciones realizadas mediante un cursor afectan a las tablas base correspondiente incluida en el conjunto de resultados del cursor.

## 5.3 Modo en que SQL Server procesa los cursores

Al acceder a los datos mediante cursores, SQL Server divide el proceso en las siguientes operaciones:

- Declaración del cursor, define los atributos de un cursor de servidor de Transact-SQL, como su comportamiento de desplazamiento y la consulta utilizada para generar el conjunto de resultados sobre el que opera el cursor.
- Apertura del cursor SQL Server ejecuta el plan de consulta. Realiza un barrido de las tablas base (en la medida en que sea necesario, como con un **SELECT** normal) y crea el conjunto de resultados del cursor. Prepara cualquier tabla temporal generada por la consulta y asigna recursos (como memoria) para dar soporte a la estructura del cursor. También coloca el cursor delante de la primera fila de su conjunto de resultados.
- Recuperación desde el cursor SQL Server desplaza la posición del cursor hacia abajo una o más filas en su conjunto de resultados. Recupera los datos de cada fila del conjunto de resultados y almacena la posición actual, permitiendo posteriores recuperaciones hasta alcanzar el final del conjunto de resultados.

- Actualización o eliminación mediante el cursor SQL Server actualiza o elimina los datos del conjunto de resultados del cursor (y las tablas base correspondientes de las que se han derivado los datos) en la posición en que se encuentre el cursor después de una recuperación. Esta operación es opcional.
- Cierre del cursor SQL Server cierra el conjunto de resultados del cursor, quita las tablas temporales que quedan y libera los recursos del servidor retenidos para la estructura del cursor. Sin embargo, conserva el plan de consulta del cursor para poder abrirlo de nuevo.
- Desasignación del cursor SQL Server vuelca el plan de consultas de la memoria y elimina toda huella de la estructura del cursor. Es preciso volver a declarar el cursor antes de utilizarlo.

#### 5.4. Declaración y apertura de cursores

Se debe declarar un cursor para poder utilizarlo. La declaración especifica la consulta que, a su vez, define el conjunto de resultados del cursor. Se puede definir un cursor como actualizable o de solo lectura de forma explícita mediante las palabras clave **for update** o **for read only**. En caso de omitirlas, SQL Server determina si el cursor es actualizable, basándose en el tipo de consulta que define su conjunto de resultados. No es posible utilizar las instrucciones **update** o **delete** con el conjunto de resultados de un cursor de solo lectura.

La sintaxis de la instrucción **declare cursor** es:

```
DECLARE cursor_name CURSOR
for select_statement
[for {read only | update [of column_name_list]]
```

La instrucción **DECLARE CURSOR** debe preceder a cualquier instrucción **OPEN** del cursor. No es posible combinar **DECLARE CURSOR** con otras instrucciones del mismo lote Transact-SQL, excepto al usar el cursor en un procedimiento almacenado.

`select_statement` es la consulta que define el conjunto de resultados del cursor para `cursor_name`. En general, `select_statement` puede utilizar la sintaxis y semántica completas de una instrucción **SELECT** Transact-SQL, incluida la palabra clave **HOLDLOCK**. Sin embargo, no puede contener una cláusula **COMPUTE**, **FOR BROWSE** o **INTO**.

En este ejemplo, se declara un cursor sencillo para las filas de la tabla Suppliers cuyo nombre de compañía empieza por 'A'.

```
DECLARE employee_cursor CURSOR FOR
SELECT CompanyName, ContactName FROM Suppliers
WHERE CompanyName LIKE 'A%'
ORDER BY CompanyName
```

### Alcance del cursor

Un cursor viene definido por su **alcance**, que determina la región en que está presente el cursor. Una vez que el alcance del cursor deja de existir, también desaparece su nombre. El alcance de los cursores viene definido por las siguientes regiones:

- Sesión: esta región comienza cuando un cliente se conecta a SQL Server y termina al desconectarse. Es distinta de las regiones definidas por procedimientos almacenados o disparadores.
- Procedimiento almacenado: esta región se inicia cuando un procedimiento almacenado comienza su ejecución y finaliza cuando la termina. Si un procedimiento almacenado llama a otro, SQL Server inicia una región nueva y la trata como subregión del primer procedimiento.
- Disparador: esta región se inicia cuando un disparador comienza su ejecución y finaliza cuando la completa.

### Conversión de los cursores en actualizables

Se puede actualizar o eliminar una fila devuelta por un cursor si este es actualizable. Si el cursor es de solo lectura, sólo podrá leer los datos, pero no actualizarlos ni eliminarlos. De forma predeterminada, SQL Server intenta determinar si un cursor es actualizable antes de designarlo como de solo lectura.

Se puede especificar si un cursor es actualizable o no, utilizando las palabras clave **read only** o **update** en la instrucción **declare**. El siguiente ejemplo define un conjunto de resultados actualizable para el cursor **suppliers\_cursor**:

```
DECLARE suppliers_cursor CURSOR FOR
    SELECT CompanyName, ContactName, Phone, Fax
    FROM Suppliers
    WHERE CompanyName LIKE 'N%'
    ORDER BY CompanyName
    FOR UPDATE OF Phone, Fax
```

El ejemplo anterior incluye el CompanyName, ContactName, Phone y Fax de la tabla Suppliers, pero solo puede actualizar las columnas Phone y Fax.

A menos que se planee actualizar o eliminar filas mediante un cursor, se debe declarar este como de solo lectura. Si no especifica **READ ONLY** o **UPDATE** explícitamente, el cursor es actualizable implícitamente cuando la instrucción **SELECT** no contiene ninguno de los siguientes elementos:

- Opción **DISTINCT**
- Cláusula **GROUP BY**
- Función agregada
- Subconsulta
- Operador **UNION**
- Cláusula **AT ISOLATION READ UNCOMMITTED**



## 5.5 Apertura de cursores

Después de declarar el cursor, se debe abrir este para realizar una operación **FETCH**, **UPDATE** o **DELETE** de las filas. La apertura de un cursor hace que SQL Server evalúe la instrucción **SELECT** que define el cursor y deje disponible el conjunto de resultados del cursor para su procesamiento. La sintaxis de **OPEN** es:

```
OPEN cursor_name
```

Después de abrir un cursor, este se coloca delante de la primera fila de su conjunto de resultados. Se debe utilizar **FETCH** para acceder a la primera fila.

SQL Server no permite abrir un cursor si este ya está abierto o no se ha definido con la instrucción **DECLARE CURSOR**, pero se puede volver a abrir un cursor cerrado para restablecer su posición al principio del conjunto de resultados.

## 5.6 Obtención de datos por medio de cursores

Después de declarar y abrir un cursor, se puede recobrar filas de un conjunto de resultados con el comando **FETCH**. Este comando devuelve una o más filas al cliente que está extrayendo los datos de columna de la fila. Si se desea, se pueden incluir parámetros de Transact-SQL o variables locales con **FETCH** para almacenar valores de columna.

La sintaxis de la instrucción **fetch** es:

```
Fetch cursor_name [into fetch_target_list]
```

Cada **FETCH** posterior recupera otra fila del conjunto de resultados del cursor.

Después de recobrar todas las filas, el cursor apuntará a la última fila del conjunto de resultados. Si se ejecuta **FETCH** de nuevo, SQL Server devuelve una advertencia a través de la variable **@@SQLSTATUS** (descrita más abajo) indicando que no hay más datos. La posición del cursor no se altera.

No es posible recobrar una fila que ya se ha recuperado, ni volver atrás en un conjunto de resultados del cursor. Se puede cerrar y reabrir el cursor para generar el conjunto de resultados del cursor nuevamente e iniciar la recuperación desde el principio.

La cláusula **INTO** especifica que SQL Server introduce datos de columna en las variables indicadas. La *fetch\_target\_list* debe componerse de parámetros de Transact-SQL o variables locales declarados anteriormente.

Ejemplo de un cursor que emite un listado tipo reporte, con la información de las personas.

```
-- DECLARANDO UN CURSOR
DECLARE Cursor_Employees CURSOR FOR
    SELECT EmployeeID, FirstName, LastName
    FROM Employees

-- IMPRESIÓN DEL TÍTULO DEL INFORME
PRINT ' Employee ID' + SPACE(5) + 'Fist Name' + SPACE(20) + 'LastName'
PRINT REPLICATE('=',80)
```

```

-- ABRIENDO EL CURSOR
OPEN Cursor_Employees
DECLARE @EmployeeID    INT,
        @FirstName     VARCHAR(10),
        @LastName      VARCHAR(20),
        @count          INT

SET @count = 0
SET NOCOUNT ON

-- LEYENDO EL CURSOR
FETCH NEXT FROM Cursor_Employees
INTO @EmployeeID, @FirstName, @LastName

WHILE @@FETCH_STATUS = 0
BEGIN
    SET @count = @count + 1

    PRINT RIGHT('00000' + CAST(@EmployeeID as varchar), 5) +
          SPACE(12) + @FirstName + SPACE(29 - LEN(@FirstName)) +
          @LastName

    FETCH NEXT FROM Cursor_Employees
    INTO @EmployeeID, @FirstName, @LastName
END

CLOSE Cursor_Employees
DEALLOCATE Cursor_Employees

--IMPRESIÓN DEL PIE DEL INFORME
PRINT REPLICATE ('=', 80)
PRINT 'TOTAL DE CLIENTES =====> ' + CONVERT(VARCHAR, @count)

```

#### A. Verificación del estado del cursor

SQL Server devuelve un valor de estado después de cada recuperación. Se puede acceder al valor mediante la variable global **@@FETCH\_STATUS**. La siguiente tabla enumera valores de **@@FETCH\_STATUS** posibles y su significado:

Valores de @@FETCH_STATUS	
Valor	Significado
0	La instrucción <b>FETCH</b> se ejecutó correctamente.
-1	La instrucción <b>FETCH</b> no se ejecutó correctamente o la fila estaba más allá del conjunto de resultados.
-2	Falta la fila capturada.

#### B. Verificación del número de filas recuperadas

SQL Server también proporciona la variable global **@@rowcount**. **@@rowcount** permite controlar el número de filas del conjunto de resultados del cursor devueltas al cliente hasta la última recuperación. En otras palabras, representa el número total de filas vistas por el cursor en un momento dado.

Una vez leídas todas las filas de un conjunto de resultados del cursor, **@@rowcount** representa el número total de filas de dicho conjunto de resultados. Cada cursor abierto está asociado a una variable **@@rowcount** específica. Esta se omite al cerrar el cursor. Verificando **@@rowcount** después

de una operación de **FETCH**, se obtendrá el número de filas leídas con el cursor especificado en dicha operación **FETCH**.

## 5.7 Eliminación y actualización de datos mediante cursores

Si el cursor es actualizable, se pueden utilizar las instrucciones **UPDATE** y **DELETE** para actualizar o eliminar filas. SQL Server determina si el cursor es actualizable verificando la *select\_statement* que define el cursor. También puede definir un cursor como actualizable de forma explícita con la cláusula **FOR UPDATE** de la instrucción **DECLARE CURSOR**.

### 5.7.1 Eliminación de filas del conjunto de resultados del cursor

Mediante la cláusula **WHERE CURRENT OF** de la instrucción **DELETE**, puede eliminar la fila de la posición actual del cursor. Al eliminar una fila del conjunto de resultados del cursor, también se elimina de la tabla de base de datos subyacente. Solo es posible eliminar una fila cada vez con el cursor.

La sintaxis de **delete...where current of** es:

```
DELETE [FROM] [[DATABASE.]OWNER.]{TABLE_NAME | VIEW_NAME}  
WHERE CURRENT OF CURSOR_NAME
```

El *table\_name* o *view\_name* especificado con **DELETE... WHERE CURRENT OF** debe ser la tabla o vista, especificada en la primera cláusula **FROM** de la instrucción **SELECT** que define el cursor.

Por ejemplo, se puede eliminar la fila a la que apunta actualmente el cursor **contact\_cursor** de la siguiente manera:

```
DELETE FROM Suppliers  
WHERE CURRENT OF suppliers_cursor
```

La palabra clave **FROM** del ejemplo anterior es opcional.

**Nota:** no es posible eliminar una fila de un cursor definido por una instrucción **SELECT** que contenga una combinación, aunque el cursor sea actualizable.

Después de eliminar una fila de un cursor, SQL Server coloca el cursor delante de la siguiente fila de su conjunto de resultados. Todavía se debe utilizar **FETCH** para tener acceso a la fila siguiente. Si se elimina la última fila del conjunto de resultados del cursor, SQL Server coloca el cursor después de la última fila del conjunto de resultados.

### 5.7.2 Actualización de filas del conjunto de resultados del cursor

Mediante la cláusula **WHERE CURRENT OF** de la instrucción **UPDATE**, se puede actualizar la fila de la posición actual del cursor. Cualquier actualización del conjunto de resultados del cursor también afecta a la fila de la tabla base de la que se deriva la fila del cursor.

La sintaxis de **UPDATE...WHERE CURRENT OF** es:

```
UPDATE [[DATABASE.]OWNER.] {TABLE_NAME | VIEW_NAME}
SET [[[[DATABASE.] OWNER.] {TABLE_NAME. | VIEW_NAME.}]]
    COLUMN_NAME 1 =
    {EXPRESSION 1 | NULL | (SELECT_STATEMENT)}
[, COLUMN_NAME 2 =
    {EXPRESSION 2 | NULL | (SELECT_STATEMENT)}] ...
WHERE CURRENT OF CURSOR_NAME
```

La cláusula **SET** especifica el nombre de columna del conjunto de resultados del cursor y asigna el valor nuevo. Cuando se enumeran varios pares del tipo nombre de columna/valor, deben ir separados por comas.

table\_name o view\_name debe ser la tabla o vista especificada en la primera cláusula **FROM** de la instrucción **SELECT** que define el cursor. Si dicha cláusula **FROM** hace referencia a más de una tabla o vista (mediante una combinación), solo se podrá especificar la tabla o vista que esté actualizando en ese momento.

Por ejemplo, se puede actualizar la fila a la que apunta el cursor **suppliers\_cursor** del siguiente modo:

```
UPDATE Suppliers
SET Phone = '(01)' + Phone,
    Fax   = '(01)' + Fax
WHERE CURRENT OF suppliers_cursor
```

Después de la actualización, la posición del cursor permanece igual. Se puede continuar actualizando la fila de esa posición del cursor siempre que otra instrucción Transact-SQL no lo desplace.

SQL Server permite actualizar columnas que no están especificadas en la lista de columnas de la select\_statement, pero forman parte de las tablas indicadas en esta instrucción. Sin embargo, cuando especifica una column\_name\_list con **UPDATE**, solo es posible actualizar las columnas indicadas en ella.

## 5.8 Cierre y liberación de memoria de cursores

Cuando se termine con el conjunto de resultados del cursor, se podrá cerrar mediante **CLOSE**. La sintaxis de **CLOSE** es:

```
CLOSE CURSOR CURSOR_NAME
```

El cierre del cursor no cambia su definición. Se puede volver a abrir con **OPEN** y SQL Server creará un conjunto de resultados del cursor nuevo utilizando la misma consulta que antes. Por ejemplo:

```
CLOSE CURSOR suppliers_cursor  
OPEN suppliers_cursor
```

Después, se puede recobrar desde **suppliers\_cursor**, empezando por el principio de un conjunto de resultados. Cualquier condición asociada a dicho cursor (como el número de filas recobradas, definido por **SET CURSOR ROWS**) permanece en vigor.

Si se quiere desechar el cursor, se debe desasignar mediante **DEALLOCATE**. La sintaxis de **DEALLOCATE** es:

```
DEALLOCATE CURSOR CURSOR_NAME
```

La liberación de memoria de un cursor es independizar cualquier recurso asociado con él, incluido el nombre del cursor. No se puede volver a utilizar un nombre de cursor hasta que lo desasigne. Si se desasigna un cursor abierto, SQL Server lo cierra automáticamente. La finalización de una conexión cliente a un servidor también cierra y desasigna cualquier cursor abierto.