

MANUALITO MS-SQL SERVER

Contenido

1. Crear Store Procedures en MS SQL Server.....	1
2. Crear Triggers en MS SQL Server.....	5
3. Crear Vistas en MS SQL Server	9

1. Crear Store Procedures en MS SQL Server

- Si lo hacemos por el Enterprise Manager, encima de la base de datos, desplegaremos la carpeta de storeds, botón derecho y "New Stored Procedure"
- El Enterprise Manager por defecto pone:
`CREATE PROCEDURE [OWNER].[PROCEDURE NAME] AS`
- Un store procedure o procedimiento almacenado es un programa dentro de la base de datos que ejecuta una acción o conjunto de acciones específicas.
- Un procedimiento tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código.
- Los procedimientos almacenados pueden devolver valores (numérico entero) o conjuntos de resultados.
- Para crear un procedimiento almacenado debemos emplear la sentencia CREATE PROCEDURE.

```
CREATE PROCEDURE <nombre_procedure> [@param1 <tipo>, ...]
```

```
AS
```

```
-- Sentencias del procedure
```

- Para modificar un procedimiento almacenado debemos emplear la sentencia ALTER PROCEDURE.

```
ALTER PROCEDURE <nombre_procedure> [@param1 <tipo>, ...]
```

AS

-- Sentencias del procedure

- El siguiente ejemplo muestra un procedimiento almacenado, denominado spu_addCliente que inserta un registro en la tabla "CLIENTES".

```
CREATE PROCEDURE spu_addCliente @nombre varchar(100),@apellido1 varchar(100),@apellido2 varchar(100),@codClivarchar(20),@fxNacimiento datetime
```

AS

```
INSERT INTO CLIENTES(nombre, apellido1, apellido2, codcli, fxnacimiento) VALUES (@nombre,  
@apellido1, @apellido2, @codcli, @fxNacimiento)
```

- Para ejecutar un procedimiento almacenado debemos utilizar la sentencia EXEC. Cuando la ejecución del procedimiento almacenado es la primera instrucción del lote, podemos omitir el uso de EXEC.
- El siguiente ejemplo muestra la ejecución del procedimiento almacenado anterior.

```
DECLARE @fecha_nacimiento datetime
```

```
set @fecha_nacimiento = convert(datetime, '29/12/1976', 103)
```

```
EXEC spu_addCliente 'David', 'Sarmiento', 'Cervantes', '00000002323', @fecha_nacimiento
```

- Siempre es deseable que las instrucciones del procedure estén dentro de un bloque TRY CATCH y controlados por una transacción.

```
ALTER PROCEDURE spu_addCliente @nombre varchar(100),@apellido1 varchar(100),@apellido2 varchar(100),@codClivarchar(20),@fxNacimiento datetime
```

AS

```
BEGIN TRY
```

```
BEGIN TRAN
```

```
INSERT INTO CLIENTES(nombre, apellido1, apellido2, codcli, fxnacimiento) VALUES(@nombre, @apellido1,  
@apellido2, @codcli, @fxNacimiento)
```

```
COMMIT
```

```
END TRY
```

```
BEGIN CATCH
```

```
ROLLBACK
```

```
PRINT ERROR_MESSAGE()
```

```
END CATCH
```

- Si queremos que los parámetros de un procedimiento almacenado sean de entrada-salida debemos especificarlo a través de la palabra clave OUTPUT, tanto en la definición del procedure como en la ejecución.
- El siguiente ejemplo muestra la definición de un procedure con parámetros de salida.

```
CREATE PROCEDURE spu_ObtenerSaldoCuenta @numCuenta varchar(20), @saldo decimal(10,2) output
```

```
AS
```

```
BEGIN
```

```
SELECT @saldo = SALDO
```

```
FROM CUENTAS
```

```
WHERE NUMCUENTA = @numCuenta
```

```
END
```

- Y para ejecutar este procedure:

```
DECLARE @saldo decimal(10,2)
```

```
EXEC spu_ObtenerSaldoCuenta '200700000001', @saldo output
```

```
PRINT @saldo
```

- Un procedimiento almacenado puede devolver valores numéricos enteros a través de la instrucción RETURN. Normalmente debemos utilizar los valores de retorno para determinar si la ejecución del procedimiento ha sido correcta o no. Si queremos obtener valores se recomienda utilizar parámetros de salida o funciones escalares.
- El siguiente ejemplo muestra un procedimiento almacenado que devuelve valores.

```
CREATE PROCEDURE spu_EstaEnNumerosRojos @numCuenta varchar(20) AS
```

```
BEGIN
```

```
IF (SELECT SALDO FROM CUENTAS
    WHERE NUMCUENTA = @numCuenta) < 0
BEGIN
    RETURN 1
END
ELSE
    RETURN 0
END
```

- El siguiente ejemplo muestra como ejecutar el procedure y obtener el valor devuelto.

```
DECLARE @rv int
EXEC @rv = spu_EstaEnNumerosRojos '200700000001'
PRINT @rv
```

- Otra característica interesante de los procedimientos almacenados es que pueden devolver uno o varios conjuntos de resultados.
- El siguiente ejemplo muestra un procedimiento almacenado que devuelve un conjunto de resultados.

```
CREATE PROCEDURE spu_MovimientosCuenta @numCuenta varchar(20) AS
BEGIN
    SELECT @numCuenta, SALDO_ANTERIOR, SALDO_POSTERIOR, IMPORTE, FXMOVIMIENTO
    FROM MOVIMIENTOS
    INNER JOIN CUENTAS ON MOVIMIENTOS.IDCUENTA = CUENTAS.IDCUENTA
    WHERE NUMCUENTA = @numCuenta
    ORDER BY FXMOVIMIENTO DESC
END
```

- La ejecución del procedimiento se realiza normalmente.

```
EXEC spu_MovimientosCuenta '200700000001'
```

- El resultado de la ejecución:

NUMCUENTA	SALDO_ANTERIOR	SALDO_POSTERIOR	IMPORTE	FXMOVIMIENTO
-----	-----	-----	-----	-----
200700000001	50.99	100.99	50.00	2007-08-25 16:18:36.490
200700000001	0.99	50.99	50.00	2007-08-23 16:20:41.183
200700000001	50.99	0.99	50.00	2007-08-23 16:16:29.840
200700000001 0.99	50.99	50.00	2007-08-23 16:14:05.900	

2. Crear Triggers en MS SQL Server

- Un trigger (o desencadenador) es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos.
- SQL Server proporciona los siguientes tipos de triggers:
 - *Trigger DML*, se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.
 - *Trigger DDL*, se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

Trigger DML.

- Los trigger DML se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.
- La sintaxis general de un trigger es la siguiente:

```
CREATE TRIGGER <Trigger_Name, sysname, Trigger_Name>

ON <Table_Name, sysname, Table_Name>

AFTER <Data_Modification_Statements, , INSERT,DELETE,UPDATE> AS

BEGIN

-- SET NOCOUNT ON added to prevent extra result sets from

-- interfering with SELECT statements.

SET NOCOUNT ON;

-- Insert statements for trigger here
```

END

- Antes de ver un ejemplo es necesario conocer las tablas inserted y deleted.
- Las instrucciones de triggers DML utilizan dos tablas especiales denominadas inserted y deleted. SQL Server 2005 crea y administra automáticamente ambas tablas. La estructura de las tablas inserted y deleted es la misma que tiene la tabla que ha desencadenado la ejecución del trigger.
- La primera tabla (inserted) solo está disponible en las operaciones INSERT y UPDATE y en ella están los valores resultantes después de la inserción o actualización. Es decir, los datos insertados. Inserted estará vacía en una operación DELETE.
- En la segunda (deleted), disponible en las operaciones UPDATE y DELETE, están los valores anteriores a la ejecución de la actualización o borrado. Es decir, los datos que serán borrados. Deleted estará vacía en una operación INSERT.
- ¿No existe una tabla UPDATED? No, hacer una actualización es lo mismo que borrar (deleted) e insertar los nuevos (inserted). La sentencia UPDATE es la única en la que inserted y deleted tienen datos simultáneamente.
- No se puede modificar directamente los datos de estas tablas.
- El siguiente ejemplo, graba un histórico de saldos cada vez que se modifica un saldo de la tabla cuentas:

```
CREATE TRIGGER TR_CUENTAS

ON CUENTAS

AFTER UPDATE

AS

BEGIN

-- SET NOCOUNT ON impide que se generen mensajes de texto

-- con cada instrucción

SET NOCOUNT ON;

INSERT INTO HCO_SALDOS(IDCuenta, SALDO, FXSALDO) SELECT IDCuenta, SALDO, getdate()FROM
INSERTED

END
```

- La siguiente instrucción provocará que el trigger se ejecute:

```
UPDATE CUENTAS

SET SALDO = SALDO + 10

WHERE IDCuenta = 1
```

- Una consideración a tener en cuenta es que el trigger se ejecutará aunque la instrucción DML (UPDATE, INSERT o DELETE) no haya afectado a ninguna fila. En este caso inserted y deleted devolverán un conjunto de datos vacío.
- Podemos especificar a qué columnas de la tabla debe afectar el trigger.

```
ALTER TRIGGER TR_CUENTAS

ON CUENTAS

AFTER UPDATE

AS

BEGIN

-- SET NOCOUNT ON impide que se generen mensajes de texto

-- con cada instrucción

SET NOCOUNT ON;

IF UPDATE(SALDO) -- Solo si se actualiza SALDO

BEGIN

INSERT INTO HCO_SALDOS(IDCuenta, SALDO, FXSALDO) SELECT IDCuenta, SALDO, getdate() FROM
INSERTED

END

END
```

- Los trigger están dentro de la transacción original (Insert, Delete o Update) por lo cual si dentro de nuestro trigger hacemos un RollBack Tran, no solo estaremos echando atrás nuestro trigger sino también toda la transacción; en otras palabras si en un trigger ponemos un RollBack Tran, la transacción de Insert, Delete o Update volverá toda hacia atrás.

```
ALTER TRIGGER TR_CUENTAS

ON CUENTAS

AFTER UPDATE AS

BEGIN
```

```
-- SET NOCOUNT ON impide que se generen mensajes de texto

-- con cada instrucción

SET NOCOUNT ON;

INSERT INTO HCO_SALDOS (IDCUENTA, SALDO, FXSALDO) SELECT IDCUENTA, SALDO, getdate() FROM
INSERTED

ROLLBACK

END
```

- En este caso obtendremos el siguiente mensaje de error:

La transacción terminó en el desencadenador. Se anuló el lote.

- Podemos activar y desactivar Triggers a través de las siguientes instrucciones.

```
-- Desactiva el trigger TR_CUENTAS

DISABLE TRIGGER TR_CUENTAS ON CUENTAS

GO

-- activa el trigger TR_CUENTAS

ENABLE TRIGGER TR_CUENTAS ON CUENTAS

GO

-- Desactiva todos los trigger de la tabla CUENTAS

ALTER TABLE CUENTAS DISABLE TRIGGER ALL

GO

-- Activa todos los trigger de la tabla CUENTAS

ALTER TABLE CUENTAS ENABLE TRIGGER ALL
```


Trigger DDL

- Los trigger DDL se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.
- La sintaxis general de un trigger es la siguiente:

```
CREATE TRIGGER <trigger_name, sysname, table_alter_drop_safety>  
  
ON DATABASE  
  
FOR <data_definition_statements, , DROP_TABLE, ALTER_TABLE> AS  
  
BEGIN  
  
    ...  
  
END
```

- La siguiente instrucción impide que se ejecuten sentencias DROP TABLE y ALTER TABLE en la base de datos.

```
CREATE TRIGGER TR_SEGURIDAD  
  
ON DATABASE FOR DROP_TABLE, ALTER_TABLE AS  
  
BEGIN  
  
RAISERROR ('No está permitido borrar ni modificar tablas !' , 16, 1)  
  
ROLLBACK TRANSACTION  
  
END
```

3. Crear Vistas en MS SQL Server

- En el modelo de datos relacional la forma de guardar la información no es la mejor para ver los datos.
- Una vista es una consulta, que refleja el contenido de una o más tablas, desde la que se puede acceder a los datos como si fuera una tabla.
- Dos son las principales razones por las que podemos crear vistas.
 - Seguridad, nos pueden interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla.
 - Comodidad, como hemos dicho el modelo relacional no es el más cómodo para visualizar los datos, lo que nos puede llevar a tener que escribir complejas sentencias SQL, tener una vista nos simplifica esta tarea.
- Las vistas no tienen una copia física de los datos, son consultas a los datos que hay en las tablas, por lo que si actualizamos los datos de una vista, estamos actualizando realmente la tabla, y si actualizamos la tabla estos cambios serán visibles desde la vista.

- **Nota:** No siempre podremos actualizar los datos de una vista, dependerá de la complejidad de la misma (dependerá de si el conjunto de resultados tiene acceso a la clave principal de la tabla o no), y del gestor de base de datos. No todos los gestores de bases de datos permiten actualizar vistas, ORACLE, por ejemplo, no lo permite, mientras que SQL Server sí.
- Para crear una vista debemos utilizar la sentencia CREATE VIEW, debiendo proporcionar un nombre a la vista y una sentencia SQL SELECT válida.

```
CREATE VIEW <nombre_vista> AS (<sentencia_select>);
```

- **Ejemplo:** Crear una vista sobre nuestra tabla alquileres, en la que se nos muestre el nombre y apellidos del cliente en lugar de su código.

```
CREATE VIEW vAlquileres AS ( SELECT nombre, apellidos, matricula  
  
FROM tAlquileres, tClientes  
  
WHERE ( tAlquileres.codigo_cliente = tClientes.codigo ) )
```

- Si queremos, modificar la definición de nuestra vista podemos utilizar la sentencia ALTER VIEW, de forma muy parecida a como lo hacíamos con las tablas. En este caso queremos añadir los campos fx_alquiler y fx_devolucion a la vista.

```
ALTER VIEW vAlquileres AS (  
  
SELECT nombre, apellidos, matricula, fx_alquiler, fx_devolucion  
  
FROM tAlquileres, tClientes  
  
WHERE ( tAlquileres.codigo_cliente = tClientes.codigo ) )
```

- Por último podemos eliminar la vista a través de la sentencia DROP VIEW. Para eliminar la vista que hemos creado anteriormente se utilizaría:

```
DROP VIEW vAlquileres;
```

- Una vista se consulta como si fuese una tabla.

Para mayores referencias sobre MS SQL Server consultar:

<http://technet.microsoft.com/es-es/library/default.aspx>