TRIGGERS Y PROCEDIMIENTO ALMACENADO

TRIGGER
Un trigger (o disparador) en una Base de datos , es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación. Dependiendo de la base de datos, los triggers pueden ser de inserción (INSERT), actualización (UPDATE) o borrado (DELETE). Algunas bases de datos pueden ejecutar triggers al crear, borrar o editar usuarios, tablas, bases de datos u otros objetos.
Contenido
Usos
Son usados para mejorar la administración de la Base de datos, sin necesidad de contar con que el usuario ejecute la sentencia de SQL.
Además, pueden generar valores de columnas, previene errores de datos, sincroniza tablas, modifica valores de una vista, etc.
Permite implementar programas basados en paradigma lógico (sistemas expertos, deducción).
[editar]Componentes principales
La estructura básica de un trigger es:
" Llamada de activación: es la sentencia que permite "disparar" el código a ejecutar.
" Restricción: es la condición necesaria para realizar el código. Esta restricción puede ser de tipo condicional o de tipo nulidad.
" Acción a ejecutar: es la secuencia de instrucciones a ejecutar una vez que se han cumplido las condiciones iniciales.
[editar]Tipos
Existen dos tipos de disparadores que se clasifican según la cantidad de ejecuciones a realizar:
" Row Triggers (o Disparadores de fila): son aquellas que se ejecutaran n-veces si se llama n-veces desde la tabla asociada al trigger
" Statement Triggers (o Disparadores de secuencia): son aquellos que sin importar la cantidad de veces que se cumpla con la condición, su ejecución es única.
Pueden ser de sesión y almacenados; pero no son de fiar[cita requerida].
[editar]Efectos y características
" No aceptan parámetros o argumentos (pero podrían almacenar los datos afectados en tablas temporales)
" No pueden ejecutar las operaciones COMMIT o ROLLBACK por que estas son parte de la sentencia SQL del disparador (únicamente a través de transacciones autónomas)
" Pueden causar errores de mutaciones en las tablas, si se han escrito de manera deficiente.
[editar]Ejemplo
Un sencillo ejemplo (para SQL Server) sería crear un Trigger para insertar un pedido de algún producto cuando la cantidad de éste, en nuestro almacén, sea inferior a un valor dado.

```
BEFORE UPDATE ON tabla_almacen
FOR ALL records
   IF :NEW.producto < 100 THEN
       INSERT INTO tabla_pedidos(producto) VALUES ('1000');
   END IF;
SELECT DBO.POLVE.TEST
END
```

Disparadores en MySQL
Los disparadores son soportados en MySQL a partir de la versión 5.0.2. Algunos de los soportes existentes son los disparadores para las sentencias INSERT, UPDATE y DELETE
El estándar SQL:2003 requiere que los disparadores den a los programadores acceso a las variables de un registro utilizando una sintaxis como REFERENCING NEW AS n. Por ejemplo, si un disparador está monitoreando los cambios en la columna salario, podría escribirse un disparador como:

```
  CREATE TRIGGER ver_salario
    BEFORE UPDATE ON empleados
    REFERENCING NEW ROW AS n, OLD ROW AS o
    FOR EACH ROW
    IF n.salario <> o.salario THEN
    END IF;
```

Como en MySQL las sentencias se ejecutan luego de escribir el signo punto y coma (;), cabe destacar que para crear un disparador en MySQL, antes se escribe la sentencia DELIMITER seguida de un carácter tal como |, la cual asigna la función del punto y coma (;) a otro carácter permitiendo que el disparador sea escrito usando los punto y comas sin que se ejecute mientras se escribe; después de escrito el disparador se escribe nuevamente la sentencia DELIMITER ; para asignar al punto y coma su función habitual.
Disparadores en PostgresQL

Triggers
En español se llaman o están traducidos por desencadenador son lo mismo que los Stored Procedures pero éstos se ejecutan desantendidamente y automáticamente cuando un usuario realiza una acción con la tabla de una base de datos que lleve asociado este trigger. Se pueden crear triggers para las sentencias de SQL Insert, Update y Delete.
Por ejemplo en la anterior tabla pr_usuararios podemos crear un trigger que cada vez que se inserte un nuevo registro envie un mail de aviso al webmaster del web (este ejemplo esta puesto al final de esta página).
La estructura de un trigger es:

```
Create Trigger pr_usuarios_Trigger1
On dbo.pr_usuarios
For /* Insert, Update, Delete */
As
```

Los triggers pueden incluir cualquier número y clase de instrucción de Transact-SQL.
Los podemos crear desde el Entreprise Manager del SQL como muestran las figuras 1 y 2 o desde el Visual Interdev creando un proyecto de base de datos como muestra en la figura 3 y 4.
Desde el Trigger podremos obtener los datos de la fila que se ha modificado o añadido utilizando inserted o deleted:

```
Select * from deleted
```

Limitaciones de los triggers.

- Solo se pueden aplicar a una tabla especifica, es decir, un trigger no sirve para dos o más tablas
- El trigger se crea en la base de datos que de trabajo pero desde un trigger puedes hacer referencia a otras bases de datos.
- Un Trigger devuelve resultados al programa que lo desencadena de la misma forma que un Stored Procedure aunque no es lo mas idoneo, para impedir que una instrucción de asignación devuelva un resultado se puede utilizar la sentencia SET NOCOUNT al principio del Trigger.
- Las siguientes instrucciones no se pueden utilizar en los triggers :

ALTER DATABASE CREATE DATABASE
DISK INIT DISK RESIZE
DROP DATABASE LOAD DATABASE
LOAD LOG RECONFIGURE
RESTORE DATABASE RESTORE LOG

Ejemplo.
Como ejemplo crearemos un Trigger que avise al webmaster con un mail cuando un usuario se da de alta en nuestro web, para crear el trigger solo se han de seguir los pasos arriba indicados, como muestran las imágenes y el código es el siguiente, haciendo Copy & Paste funciona.
El funcionamiento del trigger es muy sencillo, declaramos dos variables, una para el mensaje que se enviará en el mail y otra para obtener el ID del registro recién insertado y luego este ID lo concatenamos al mensaje para enviárselo al webmaster.

```
Alter Trigger Trigger_Aviso_al_Webmaster
On dbo.pr_usuarios
For Insert
As
– Declaramos las variables del mensaje y del ID del nuevo usuario
Declare @Mensaje varchar(200)
Declare @ID numeric
– Obtenemos el id del usuario recien insertado
Select @ID = (Select IDUsuario From Inserted )
Select @Mensaje = ‘Nuevo Usuarios en el web : ‘ + Convert(varchar(10), @ID)
Exec master.dbo.xp_sendmail
@recipients = ‘webmaster@dominio.com’,
@subject = ‘Nuevo usuario’,
@message = @Mensaje
```

PROCEDIMIENTO ALMACENADO
Un procedimiento almacenado (*stored procedure* en inglés) es un programa (o procedimiento) el cual es almacenado físicamente en una base de datos. Su implementación varía de un gestor de bases de datos a otro. La ventaja de un procedimiento almacenado es que al ser ejecutado, en respuesta a una petición de

usuario, es ejecutado directamente en el motor de bases de datos, el cual usualmente corre en un servidor separado. Como tal, posee acceso directo a los datos que necesita manipular y sólo necesita enviar sus resultados de regreso al usuario, deshaciéndose de la sobrecarga resultante de comunicar grandes cantidades de datos salientes y entrantes.
Usos típicos para procedimientos almacenados incluyen la validación de datos siendo integrados a la estructura de base de datos (los procedimientos almacenados utilizados para este propósito a menudo son llamados disparadores; *triggers* en inglés), o encapsular un proceso grande y complejo. El último ejemplo generalmente ejecutará más rápido como un procedimiento almacenado que de haber sido implementado como, por ejemplo, un programa corriendo en el sistema cliente y comunicándose con la base de datos mediante el envío de consultas SQL y recibiendo sus resultados.
Los procedimientos pueden ser ventajosos: Cuando una base de datos es manipulada desde muchos programas externos. Al incluir la lógica de la aplicación en la base de datos utilizando procedimientos almacenados, la necesidad de embeber la misma lógica en todos los programas que acceden a los datos es reducida. Esto puede simplificar la creación y, particularmente, el mantenimiento de los programas involucrados.
Podemos ver un claro ejemplo de estos procedimientos cuando requerimos realizar una misma operación en un servidor dentro de algunas o todas las bases de datos y a la vez dentro de todas o algunas de las tablas de las bases de datos del mismo. Para ello podemos utilizar a los Procedimientos almacenados auto creables que es una forma de generar ciclos redundantes a través de los procedimientos almacenados.

Implementación

Estos procedimientos, se usan a menudo, pero no siempre, para realizar consultas SQL sobre los objetos del banco de datos de una manera abstracta, desde el punto de vista del cliente de la aplicación.
Un procedimiento almacenado permite agrupar en forma exclusiva parte de algo específico que se desee realizar o, mejor dicho, el SQL apropiado para dicha acción. CORRECTO
Usos

Los usos 'típicos' de los procedimientos almacenados se aplican en la validación de datos, integrados dentro de la estructura del banco de datos. Losprocedimientos almacenados usados con tal propósito se llaman comúnmente disparadores, o *triggers*. Otro uso común es la 'encapsulación' de unAPI para un proceso complejo o grande que podría requerir la 'ejecución' de varias consultas SQL, tales como la manipulación de un 'dataset' enorme para producir un resultado resumido.
También pueden ser usados para el control de gestión de operaciones, y ejecutar procedimientos almacenados dentro de una transacción de tal manera que las transacciones sean efectivamente transparentes para ellos.
[editar]Ventajas

La ventaja de un procedimiento almacenado, en respuesta a una petición de usuario, está directamente bajo el control del motor del manejador de bases de datos, lo cual corre generalmente en un servidor separado de manejador de bases de datos aumentando con ello, la rapidez de procesamiento de requerimientos del manejador de bases de datos. El servidor de la base de datos tiene acceso directo a los datos necesarios para manipular y sólo necesita enviar el resultado final al usuario. Los procedimientos almacenados pueden permitir que la lógica del negocio se encuentre como un API en la base de datos, que pueden simplificar la gestión de datos y reducir la necesidad de codificar la lógica en el resto de los programas cliente. Esto puede reducir la probabilidad de que los datos sean corrompidos por el uso de programas clientes defectuosos o erróneos. De este modo, el motor de base de datos puede asegurar la integridad de los datos y la consistencia, con la ayuda de procedimientos almacenados. Algunos afirman que las bases de datos deben ser utilizadas para el almacenamiento de datos solamente, y que la lógica de negocio sólo debería ser aplicada en la capa de negocio de código, a través de aplicaciones cliente que deban acceder a los datos. Sin embargo, el uso de procedimientos almacenados no se opone a la utilización de una capa de negocio.
[editar]Procedimientos almacenados en MySQL

Desde MySQL 5 los procedimientos almacenados empezaron a ser soportados, como suele suceder en MySQL las sentencias se ejecutan luego de escribir el signo punto y coma (;), por esta razón antes de escribir el procedimiento almacenado la función del punto y coma se asigna a otros caracteres usando la sentencia *DELIMITER* seguida de un caracter tal como |, de esta manera el procedimiento puede ser escrito usando los punto y comas sin que se ejecute mientras se escribe; después de escrito el procedimiento, se escribe nuevamente la sentencia *DELIMITER ;* para asignar al punto y coma su función habitual.Fven
El siguiente es un ejemplo de procedimiento almacenado en MySQL:

```
DELIMITER |

CREATE PROCEDURE autos(IN velocidad INT,IN marca VARCHAR(50))
BEGIN
```

```
IF velocidad < 120 THEN
INSERT INTO familiares VALUES(velocidad,marca);
ELSE
INSERT INTO deportivos VALUES(velocidad,marca);
END IF;
END;
|
```

[SQL]Procedimientos almacenados paso a paso

octubre 31, 2008norrojasDeja un comentarioIr a los comentarios

Un procedimiento almacenado (store procedure) no es más que una secuencia ordenada de instrucciones T-SQL, que pueden recibir y proporcionar parámetros provistos por el usuario y se pueden guardar en el servidor con un nombre, para posteriormente se invocados y ejecutados, por lo regular desde una aplicación (Escritorio o Web). Desde la versión 2005, se incorpora la posibilidad de utilizar procedimientos almacenados usando el CLR de .NET. Es decir tenemos dos tipos de procedimientos almacenados.

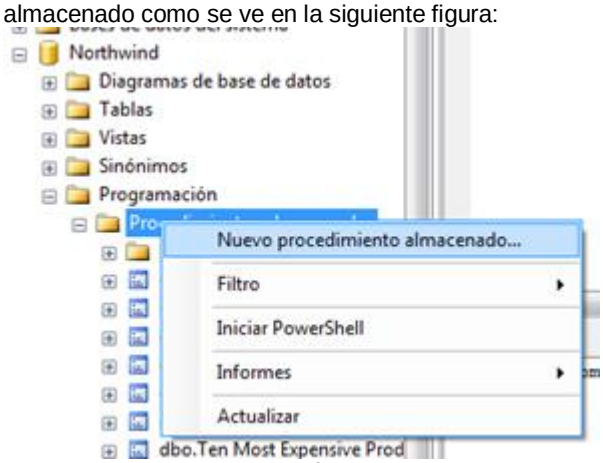
Un procedimiento almacendado CLR es una referencia a un método de un ensamble (dll) de .NET Framework que puede aceptar y devolver parámetros suministrados por el usuario.

Ventajas de usar SP

- Compilación: La primera vez que se invoca un SP, el motor lo compila y a partir de ahí, se sigue usando la versión compilada del mismo, hasta que se modifique o se reinicie el servicio de SQL. Esto siginifica que se tendrá un mejor rendimiento que las consultas directas que usan cadenas con las instrucciones T-SQL, que se compilan cada vez que se invocan.
- Automatización: si tenemos un conjunto de instrucciones T-SQL, las cuales queremos ejecutar de manera ordenada, un SP es la mejor manera de hacerlo.
- Administración: cuando realizamos aplicaciones con un gran numero de lineas de código, y queremos hacer cambios, solo implica modificar un SP y no toda la aplicación, lo que significa solo cambiamos los SP en el servidor y no tenemos que actualizar la aplicación en todos los equipos cliente.
- Seguridad: una parte importante es que a los usuarios de nuestra aplicación, solo les proporcionamos los permisos para ejecutar los procedimientos almacenados y no el acceso a todos los objetos de la base. Es decir, si en nuestra aplicación encuentran una vulnerabilidad como SLQ Injection no se podrá explotar ejecutando SQL directamente.
- Programabilidad: Los SP admiten el uso de variables y estructuras de control como IF, Bucles, Case, etc. además del manejo de transacción y permite controlar excepciones. Y cuando trabajamos con SP CLR podemos hacer uso de cualquier lenguaje .NET como lo son C# y VB.NET.
- Trafico de Red: Pueden reducir el trafico de la red, debido a que se trabaja sobre el motor (en el servidor), y si una operación incluye hacer un trabajo de lectura primero y en base a eso realizar algunas operaciones, esos datos que se obtienen no viajan por la red.

Creando un Procedimiento almacenado

Para crear un procedimiento almacenado solo basta con ir a la base de datos desplegar el árbol hasta la parte de programación y luego en procedimientos almacenados y damos clic derecho en nuevo procedimiento almacenado como se ve en la siguiente figura:



Lo cual genera el siguiente código:

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
-- ============================================
-- Author:        <Author,,Name>
-- Create date: <Create Date,,>
-- Description:    <Description,,>
-- ============================================
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
   -- Add the parameters for the stored procedure here
   <@Param1, sysname, @p1> <Datatype_For_Param1, , int> = <Default_Value_For_Param1, , 0>,
   <@Param2, sysname, @p2> <Datatype_For_Param2, , int> = <Default_Value_For_Param2, , 0>
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

   -- Insert statements for procedure here
   SELECT <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
GO
```

Es aquí en donde editamos nuestro procedimiento como mejor nos interese, en este caso usando la base de datos Northwind, crearemos un SP sencillo que queda más o menos así:

```
USE [Northwind]
GO

CREATE PROCEDURE [dbo].[GetAllEmployees]
   -- Add the parameters for the stored procedure here
   @LastName nvarchar(50),
   @FirstName nvarchar(50)
AS
BEGIN
   -- SET NOCOUNT ON added to prevent extra result sets from
   -- interfering with SELECT statements.
   SET NOCOUNT ON;

   -- Insert statements for procedure here
   SELECT  LastName, FirstName, Title
   FROM dbo.Employees
   WHERE FirstName = @FirstName AND LastName = @LastName
END
```

En este sencillo ejemplo, sólo devuelve el empleado especificado (nombre y apellidos), y el titulo a partir de una vista. Este procedimiento almacenado acepta coincidencias exactas de los parámetros pasados.
Para ejecutar el código debemos hacerlo de esta manera:

```
EXEC dbo.GetAllEmployees 'Davolio', 'Nancy'
```

Cómo lo consumimos desde una aplicación en .NET, pues de esta manera:

```
static void Main(string[] args)
      {
         //Creamos una nueva conexion.
         SqlConnection miConn = new SqlConnection("Data Source=NORTABLEPC\\SQLEXPRESS;Initial
Catalog=Northwind;Integrated Security=True");

         //Creamos un nuevo comando
         SqlCommand miComm = new SqlCommand();
         //Le asignamos la conexion.
         miComm.Connection = miConn;
         //especificamos que el comando es un stored procedure
         miComm.CommandType = System.Data.CommandType.StoredProcedure;
         //y escribimos el nombre del stored procedure a invocar
         miComm.CommandText = "dbo.GetAllEmployees";
```

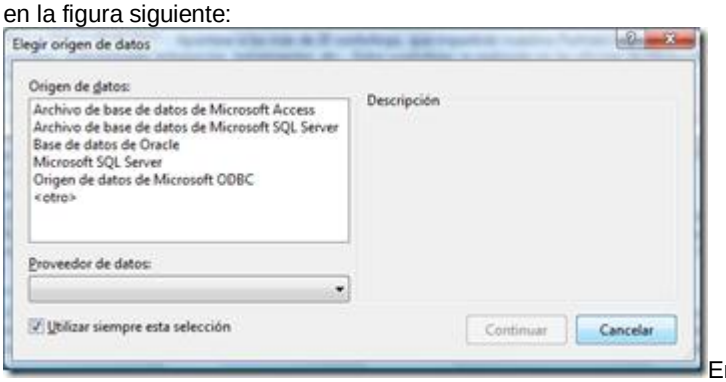
```
        //Creamos un nuevo parametro
        SqlParameter paramLastName = new SqlParameter();
        paramLastName.ParameterName = "@LastName";
        paramLastName.SqlDbType = System.Data.SqlDbType.NVarChar;
        paramLastName.Value = "Davolio";

        miComm.Parameters.Add(paramLastName);

        SqlParameter paramFirstName = new SqlParameter();
        paramFirstName.ParameterName = "@FirstName";
        paramFirstName.SqlDbType = SqlDbType.NVarChar;
        paramFirstName.Value = "Nancy";
        miComm.Parameters.Add(paramFirstName);
        //Y los agregamos a la coleccion de parametros del comando myComm.Parameters.Add(myParam)
'Creamos un nuevo DataAdapter con nuestro comando.
        SqlDataAdapter miDA = new SqlDataAdapter(miComm);
        //Creamos un dataset para soportar los datos devueltos por el stored procedure
        DataSet EmpleadosDS = new DataSet();
        //Pedimos al Data Adapter que llene el dataset (Esto llama a nuestro comando)
        miDA.Fill(EmpleadosDS);
        //Y lo mostramos por pantalla

        foreach (DataRow row in EmpleadosDS.Tables[0].Rows)
        {
            Console.WriteLine(row["LastName"].ToString() + " " + row[1].ToString() + " " + row[2].ToString());
        }
    }
```

Procedimientos Almacenados con CLR

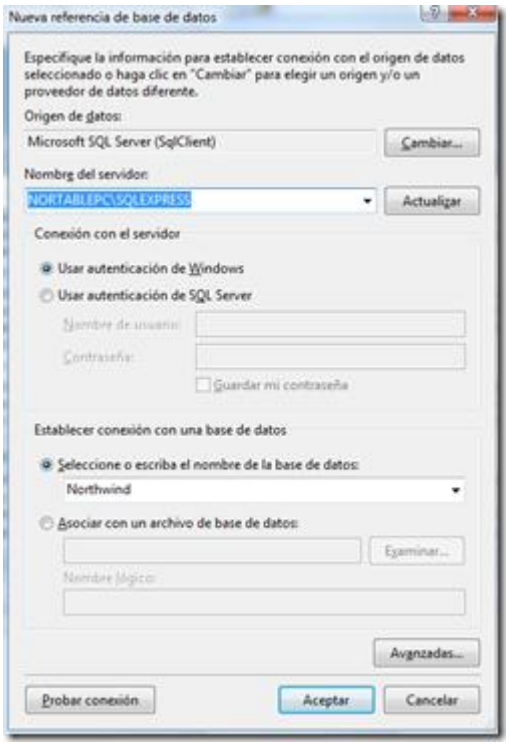
En Common Language Runtime (CLR), los procedimientos almacenados se implementan como métodos estáticos públicos en una clase de un ensamblado de Microsoft .NET Framework. El método estático se puede declarar como void o puede devolver un valor entero. Si devuelve un valor entero, éste se trata como el código devuelto desde el procedimiento. Los parámetros que se pasan a un procedimiento almacenado CLR pueden ser cualquiera de los tipos nativos de SQL Server que tengan un equivalente en código administrado. Para que la sintaxis Transact-SQL cree el procedimiento, estos tipos se deben especificar con el equivalente del tipo nativo de SQL más adecuado.

Ejemplo:

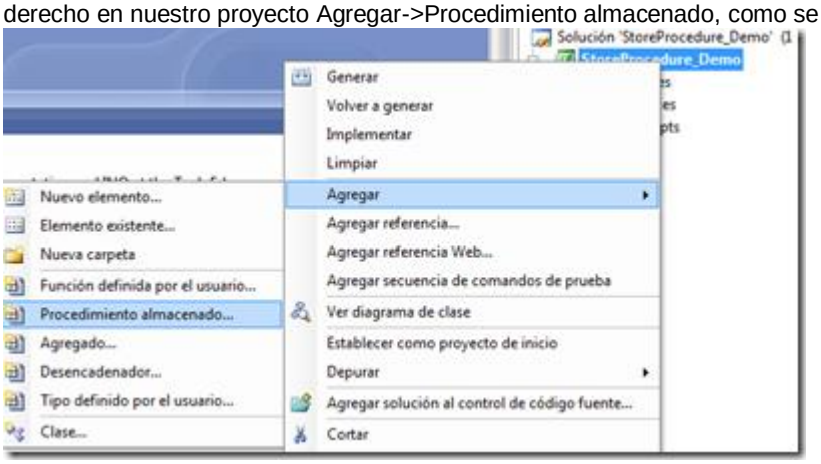
Para crear un SP con integración del CLR vamos a usar VS, y creamos un Proyecto de tipo SQL, al que llame StoreProcedure_Demo, lo primero que hace Visual Studio es preguntar el origen de datos, como se ve en la figura siguiente:



En donde seleccionamos Microsoft SQL Server (SqlClient) y damos clic en continuar. Ahora toca el turno de elegir el Servidor



En donde yo selecciono el servidor NORTABLEPC\SQLEXPRESS, y la base de datos Northwind, luego verifico la conexión y si todo va bien, por ultimo nos pedirá que si queremos habilitar la integración con el CLR, damos clic y listo. Luego damos clic derecho en nuestro proyecto Agregar->Procedimiento almacenado, como se ve en la figura:

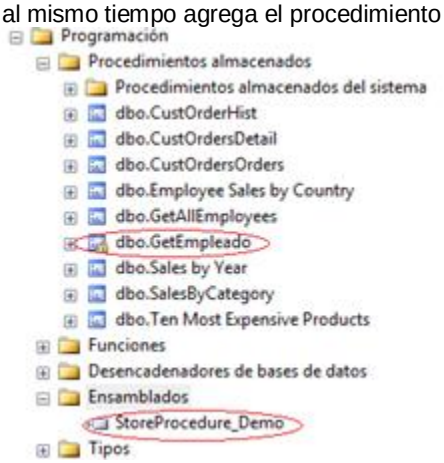


Al procedimiento lo llamamos: GetEmpleado, damos y damos clic en Aceptar. Ahora vamos a escribir el siguiente código:

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using Microsoft.SqlServer.Server;

public partial class StoredProcedures
{
    [Microsoft.SqlServer.Server.SqlProcedure (Name="GetEmpleado")]
    public static void GetEmpleado()
    {
        // Inserte el código aquí
        SqlContext.Pipe.Send("Hola Mundo! son las : " + System.DateTime.Now.ToString() + "\n");
        using (SqlConnection connection = new SqlConnection("context connection=true"))
```

```
        {
            connection.Open();
            SqlCommand command = new SqlCommand("SELECT LastName, FirstName FROM Employees",
connection);
            SqlDataReader reader = command.ExecuteReader();
            SqlContext.Pipe.Send(reader);
        }

    }
};
```

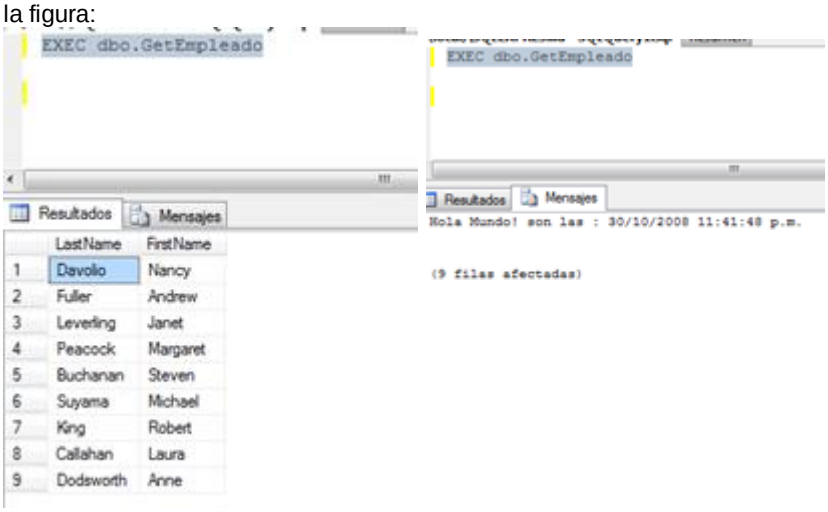
En el código anterior solo mandamos un mensaje, y devolvemos un select sencillo de los empleados, Bien para que esto funcione, debemos generar una solución de nuestro proyecto y después
una implementación del mismo, esto permite que automáticamente se envié la dll a nuestra base de datos con el nombre de nuestro proyecto, y al mismo tiempo agrega el procedimiento llamado GetEmpledo



Ahora solo nos resta ejecutar nuestro procedimiento con la siguiente instrucción:

```
EXEC dbo.GetEmpleado
```

Lo interesante de esto, es que una vez que ya hemos llamado al SP, cuando se ejecute de nuevo, ya no se compilara instrucción por instrucción solo se llama al dll y listo, el resultado de ejecutarlo es el siguiente, por un lado manda el resultado y por otro el mensaje de “hola mundo” con la hora de la maquina, como se ve en la figura:



Hasta aquí dejamos este post, ya veremos en otro como ejecutar dichos procedimientos almacenados desde una aplicación. Espero que les sea de utilidad este post y recuerden si lo creen conveniente tomen 5 min, para votar por este blog en la siguiente página :http://www.blogit.ms/ , voten por http://mspnor.wordpress.com
Saludos!!!
Etiquetas de Technorati: SQL Server,SQL Server 2005,SQL Server 2008

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 deletedes 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 despues de la inserción o actualización. Es decir, los datos insertados. Inserted estará vacia 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 vacia en una operacion 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 simultaneamente.

No puede se modificar directamente los datos de estas tablas.

El siguiente ejemplo, graba un historico 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 instruccion DML (UPDATE, INSERT o DELETE) no haya afectado a ninguna fila. En este caso inserted y deleted devolveran un conjunto de datos vacio.

Podemos especificar a que 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 tarvés 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
```