Stored Procedures, Triggers y Funciones

Stored Procedures (Procedimientos Almacenados)

Un procedimiento almacenado es un programa (o procedimiento) el cual es almacenado físicamente en una base de datos.
La **ventaja** de los procedimientos almacenados es que se ejecutan directamente en el motor de bases de datos, directo en el mismo servidor donde se encuentra ubicada la base de datos. Como tal, poseen acceso directo a los datos que necesitan manipular y sólo necesitan enviar sus resultados de regreso al usuario, deshaciéndose de la sobrecarga resultante de comunicar grandes cantidades de datos salientes y entrantes.
Generalmente se utilizan para procesos complejos o grandes que podrían requerir la 'ejecución' de varias consultas SQL, tales como la manipulación de un 'dataset' enorme para producir un resultado resumido.

Los SP tienen un nombre y se ejecutan a petición del usuario/cliente.

Creación de un SP en SQL Server

```
CREATE PROCEDURE procedure_name [ @parametro1 tipo_de_dato,
                                  @parametro2 tipo_de_dato,
                                  …
                                  @parametroN tipo_de_dato    ]
AS

[
DECLARE @variable1 tipo_de_dato;
DECLARE @variableN tipo_de_dato;
]

[BEGIN]

   Sentencias de Transact SQL

[END;]
```

Ejecución de un SP

```
EXECUTE procedure_name [parametros]
```

Desventajas:
- La lógica de la aplicación termina distribuida parte en la base de datos y parte en el código de la aplicación.
- Aumenta la dependencia del repositorio de datos (El lenguaje de los SP suele ser bastante diferente entre los distintos motores de BD, por ej., Transact-SQL para SQL Server y PL/SQL para Oracle.) Si en algún momento, se decide cambiar de Base de Datos, esa migración será más compleja.

Ventajas:
- Reutilización de código (distintos sistemas pueden utilizar los mismos SP). Por ejemplo, un mismo SP se puede disparar desde una página web, desde una aplicación Android o desde una clásica aplicación cliente-servidor.
- Mayor rendimiento (los datos no viajan desde la BD hasta la Aplicación para luego hacer cálculos, sino que todo se calcula dentro de la base y solo se entrega el resultado).
- Tráfico de Red: Pueden reducir el tráfico 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.

Ejemplo:
Dada la siguiente tabla:

Tabla DIRECTORIO

ID_DIR	NOMBRE_DIR	FECHA_CREACION	ID_DIR_PADRE
1	\	01/01/2015	
10	Programas\	05/01/2015	1
11	Programa1\	05/01/2015	10
12	Config\	05/01/2015	11
13	Data\	05/01/2015	11
14	Programa2\	06/01/2015	10
15	Log\	07/01/2015	14
20	Fotos\	02/01/2016	1
21	Viaje1\	15/01/2016	20
22	Viaje2\	10/05/2016	20
23	Viaje3\	03/06/2016	20
30	Documentos\	08/01/2015	1
31	UNLaM\	10/04/2016	30
32	Base_de_Datos\	12/04/2016	31
33	Fisica_1\	20/04/2015	31

El siguiente SP recibe como parámetro el ID de un directorio y devuelve el Path completo de su ubicación. Cómo la cantidad de niveles que puede tener ese directorio es variable, esto no se puede resolver con una simple consulta SQL (salvo que se fije una cantidad máxima de niveles).
Este SP posee una Iteración que recorre todos los niveles y arma el Path Completo concatenando los nombres de los directorios donde se encuentra ubicado.

```
CREATE PROCEDURE sp_ListaDirectorios (@pDir int) AS

DECLARE @Padre int;
DECLARE @pPathCompleto varchar(200);

BEGIN

   --Primero toma el nombre del Directorio recibido por parámetro
   SET @pPathCompleto = (SELECT nombre_dir
                         FROM   Directorio
                         WHERE  id_dir = @pDir);

   --Se fija si tiene algún directorio Padre
   SET @Padre = (SELECT isnull(id_dir_padre,0)
                 FROM   Directorio
                 WHERE  id_dir = @pDir);

   --Mientras exista algún Padre, va concatenando los nombres y armando el Path.
   WHILE (@Padre<>0) begin

         --Agrega el nombre del Padre
         SET @pPathCompleto = (SELECT nombre_dir + @pPathCompleto
                               FROM   Directorio
                               WHERE  id_dir = @Padre);

         --Se fija si tiene otro directorio Padre
         SET @Padre = (SELECT isnull(id_dir_padre,0)
                       FROM   Directorio
                       WHERE  id_dir = @Padre);
   END;

   --Finalmente, devuelve el valor
   SELECT @pPathCompleto as RESULTADO;
END;
```

Una vez creado, el Stored Procedure puede ejecutarse de la siguiente manera:

```
EXECUTE sp_ListaDirectorios 33
```

Resultado:
\Documentos\UNLaM\Fisica_1\

Triggers (Disparadores o Desencadenadores)

Es el mismo concepto de los SP pero la diferencia es que los Triggers se ejecutan por algún evento de la base de datos (por ejemplo, cuando se inserta una fila en una tabla).
Se utilizan comúnmente para hacer validaciones de datos.

Creación de un SP en SQL Server

```
CREATE TRIGGER trigger_name ON objeto FOR/AFTER/INSTEAD OF tipo_evento
AS
[
DECLARE @variable1 tipo_de_dato;
DECLARE @variableN tipo_de_dato;
]
[BEGIN]

   Sentencias de Transact SQL

[END;]
```

Objeto: Tabla / Vista / Database / ALL SERVER
Tipo de Evento: INSERT / UPDATE / DELETE / CREATE / DROP / ALTER / etc.

FOR / AFTER: El trigger se ejecuta después de que el evento se ejecutó correctamente en forma completa.
Se pueden definir varios Triggers de este tipo para un mismo objeto y un mismo evento.

INSTEAD OF: El trigger se ejecuta en vez del evento disparador (antes), por lo que se puede suplantar el evento original por otra acción.
Se puede definir un único Trigger del tipo Instead Of un mismo objeto y un mismo evento.

Tablas INSERTED y DELETED
Las tablas INSERTED y DELETED se pueden utilizar dentro del Trigger y contienen los registros insertados o eliminados. Son tablas virtuales y tienen la misma estructura (mismas columnas) que la tabla sobre la cual se define el Trigger.
No existe una tabla UPDATED, cuando se hace un UPDATE, la fila con el valor viejo queda en la tabla DELETED y la fila con el valor nuevo queda en la tabla INSERTED.

Ejemplo 1:
El siguiente Trigger incrementa en 1 la cantidad de ventas del vendedor que corresponda, cada vez que se hace una Venta y se inserta un registro en la tabla VENTAS.

```
CREATE TRIGGER ActualizaVentasVendedores
ON Ventas FOR INSERT
AS
  UPDATE      Vendedores
  SET         cant_ventas = cant_ventas + 1
  WHERE       id_vendedor IN (SELECT i.id_vendedor FROM inserted i)
```

Ejemplo 2:
Cada vez que se elimina un Producto, se guarda una copia de la fila eliminada en la tabla Producto_Eliminado que tiene exactamente la misma estructura que la tabla Producto.

```
CREATE TRIGGER t_prod_eliminado
ON Producto FOR delete
AS
   INSERT INTO Producto_Eliminado
   SELECT * FROM deleted
```

Ejemplo 3:
Cada vez que se intenta eliminar un Cliente, se acciona un Trigger antes que cancela la eliminación de la fila y simplemente hace una marca de borrado lógico en la fila y guarda también la fecha del borrado.

```
CREATE TRIGGER t_cli_eliminado
ON Cliente INSTEAD OF delete
AS
   UPDATE Cliente
   SET eliminado=1, fecha_eliminado=getdate()
   WHERE cod_cli IN (SELECT cod_cli FROM deleted)
```

Ejemplo 4:
Cada vez que se modifica la tabla Cliente, se guarda la fila Vieja (sin el cambio) y la fila Nueva (con el cambio) en una tabla de Log.

```
CREATE TRIGGER Cliente_Update_Log
ON Cliente AFTER UPDATE
AS
BEGIN

    INSERT INTO Cliente_Log
    SELECT getdate(), 'Viejo', d.*
    FROM DELETED d;

    INSERT INTO Cliente_Log
    SELECT getdate(), 'Nuevo', i.*
    FROM INSERTED i;

END;
```

Ejemplo 5:
Dadas las siguientes tablas:
Empleado (legajo, nombre, apellido)
Asignacion (idCargo, legajo, fecha_ini, fecha_fin)
Realice un trigger que solo permita borrar un empleado si no tiene ninguna Asignacion vigente (fecha_fin nula o mayor a la fecha actual), caso contrario ignore la instrucción.

```
CREATE TRIGGER Ejercicio ON Empleado INSTEAD OF DELETE AS
    DELETE Empleado
    WHERE Empleado.legajo IN (SELECT legajo FROM deleted)
    AND NOT EXISTS
    (
    SELECT  *
    FROM    Asignacion a
    WHERE   a.Legajo = Empleado.legajo
    AND     (a.fecha_fin IS NULL OR fecha_fin >= getdate() )
    );
```

Funciones

Es el mismo concepto de los SP pero con la diferencia que las funciones siempre deben retornar un valor como resultado.
Luego, se utilizan comúnmente en las sentencias SELECT.

En SQL Server existen 2 tipos de funciones:
- Funciones escalares (devuelven como resultado un valor único)
- Funciones con valores de tabla (devuelven como resultado una tabla)

Las que más se usan son las funciones escalares.

Creación de una Función escalar en SQL Server

```
CREATE FUNCTION function_name [ @parametro1 tipo_de_dato,
                                @parametro2 tipo_de_dato,
                                …
                                @parametroN tipo_de_dato    ]
RETURNS tipo_de_dato
AS
[
DECLARE @variable1 tipo_de_dato;
DECLARE @variableN tipo_de_dato;
]
[BEGIN]

   Sentencias de Transact SQL

   RETURN valor_a_retornar

[END;]
```

Ejemplo1: Función escalar

```
CREATE FUNCTION NombreMes (@nro_mes int)
RETURNS varchar(20)
AS
BEGIN
  DECLARE @nombre_mes varchar(20)

  SET @nombre_mes = (
                        CASE @nro_mes
                            WHEN 1 THEN 'Enero'
                            WHEN 2 THEN 'Febrero'
                            …
                            WHEN 12 THEN 'Diciembre'
                            ELSE 'Desconocido'
                        END
                    );

    RETURN (@nombre_mes);
END

Ejecución de la función creada
SELECT legajo, NombreMes(month(fecha_nacimiento)) Mes_Cumpleaños
FROM Empleado
```

Funciones con valor de Tabla

Dentro de este tipo de funciones existen 2 subtipos:

- Funciones con valores de tabla de varias instrucciones
- Funciones con valores de tabla en línea

Ambas retornan una tabla como resultado, pero la diferencia es que en el primer tipo se define una tabla y se carga con datos para finalmente retornarla como resultado. Mientras que el segundo caso es más sencillo y solo devuelve el resultado de una consulta.

Veamos un ejemplo de cada una:

Ejemplo 2: Función con valores de tabla de varias instrucciones
Recibe como parámetro un país y devuelve todos los clientes de ese país.

```
CREATE FUNCTION Clientes_x_Pais (@pais varchar(64))
RETURNS @clientes TABLE
 (customer_id integer
  ,razon_social varchar(50)
  ,pais varchar(64)
 )
AS
 BEGIN

 INSERT @clientes
 SELECT customer_id, razon_social, pais
 FROM   Cliente
 WHERE  pais = @pais;

 RETURN

 END

Ejecución de la función creada
SELECT * FROM Clientes_x_Pais('Chile')
```

Ejemplo 3: Función con valores de tabla en línea
La misma función que el ejemplo anterior, recibe un país como parámetro y devuelve todos los clientes de ese país.

```
CREATE FUNCTION Clientes_x_Pais2 (@pais varchar(64))
RETURNS TABLE
AS
RETURN
(
 SELECT customer_id, razon_social, pais
 FROM   Cliente
 WHERE  pais = @pais
)

Ejecución de la función creada
SELECT * FROM Clientes_x_Pais2('Chile')
```