CIBERTEC
Empieza por ti

Base de Datos Avanzado I

ÍNDICE

Página

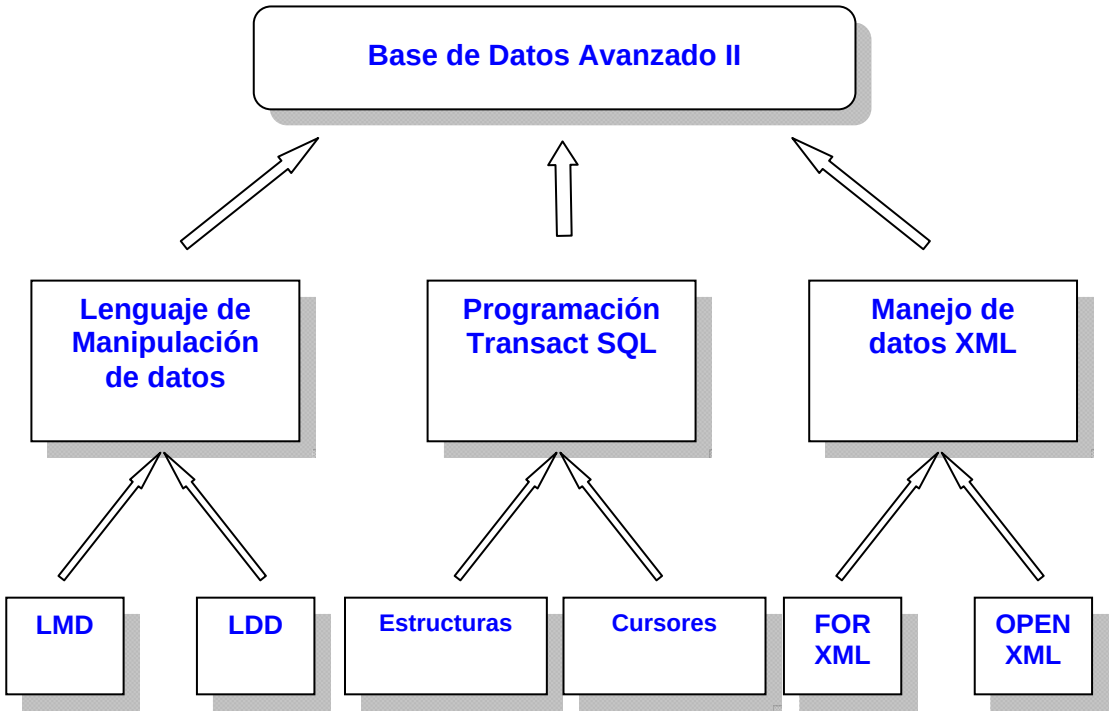
PRESENTACIÓN

Base de Datos Avanzado I es un curso que pertenece a la Escuela de Tecnologías de Información y se dicta en las carreras de Administración y Sistemas, y Computación e Informática. El presente manual ha sido desarrollado para que los alumnos del curso de Base de Datos Avanzado I puedan aplicar los conocimientos adquiridos en el curso de Base de Datos teoría y laboratorio. Todo ello, en conjunto, le permitirá manejar los datos de una base de datos relacional utilizando comandos TRANSACT-SQL.

El manual para el curso ha sido diseñado bajo la modalidad de unidades de aprendizaje, las que se desarrollan durante semanas determinadas. En cada una de ellas, hallará los logros, que debe alcanzar al final de la unidad; el tema tratado, el cual será ampliamente desarrollado; y los contenidos, que debe desarrollar, es decir, los subtemas. Por último, encontrará las actividades que deberá desarrollar en cada sesión, que le permitirán reforzar lo aprendido en la clase.

El curso es eminentemente práctico. Se inicia con la creación de la base de datos de trabajo usando el lenguaje Transact/SQL en el manejador de base de datos relacional SQL Server 2008. Posteriormente, se efectúa un repaso de las operaciones básicas de manipulación de datos (Data Manipulation Lenguaje – DML) para hacer uso de comandos que se emplean en la inserción, modificación y eliminación de los mismos. A continuación vamos a realizar operaciones de consulta avanzada de base de datos utilizando clausulas de unión, de agrupamiento, de combinación, entre otras. A continuación aprenderemos a manejar la programación TRANSACT-SQL aplicando los conceptos en cursores, procedimientos almacenados, funciones y desencadenantes o trigger. Para integrar los temas de actualidad, aprenderemos a manejar datos XML en la base de datos relacional y finalmente, en la última parte del manual, aprenderemos a manejar usuarios y generar copias de respaldo de una base de datos y restaurar una base de datos de SQL SERVER.

RED DE CONTENIDOS



UNIDAD DE APRENDIZAJE
1

LENGUAJE DE MANIPULACIÓN DE DATOS (DML)

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno recupera, inserta, actualiza y elimina información de una base de datos aplicando múltiples condiciones de comparación o funciones para el manejo de campos tipo fecha. Obtiene registros originados por la selección de uno o varios grupos haciendo uso de las funciones agrupamiento y columna procedentes de dos o más tablas.

TEMARIO

1.1. Introducción
1.1.1. Tablas a usar en el curso
1.1.2. Manejo de Esquemas
1.2. **Lenguaje para la manipulación de datos DML (3 horas)**
1.2.1. Operadores
1.2.2. Funciones usados en las consultas condicionales
    1.2.2.1. Funciones para el manejo de fecha
    1.2.2.2. Funciones pare el manejo de cadena
    1.2.2.3. Funciones de conversión de datos
1.2.3. Inserción de datos: INSERT
1.2.4. Actualización de datos: UPDATE
1.2.5. Eliminación de datos: DELETE
1.2.6. Selección de datos: SELECT
1.2.7. Instrucción MERGE
1.3. Recuperación avanzada de datos (6 horas)
1.3.1. Combinación de tablas: JOIN
1.3.2. Consultas agregadas: empleo de *GROUP BY, HAVING.* Empleo de funciones agregadas: *SUM, MIN, MAX, AVG, COUNT.*
1.3.3. Agregar conjunto de resultados: *UNION*
1.3.4. Resumen de datos: CUBE
1.3.5. Resumen de datos: ROLLUP

ACTIVIDADES PROPUESTAS

- Los alumnos implementan sentencias SQL para recuperar y actualizar datos en una base de datos relacional.
- Los alumnos implementan sentencias SQL para agrupar y resumir los datos.

1.1 INTRODUCCION

1.1.1 Estructura de la Base de Datos Negocios2011

En el curso, usaremos las tablas de la base de datos **NEGOCIOS2011**. A continuación, se muestra la estructura de algunas tablas de la base de datos **NEGOCIOS2011** a utilizar en el presente curso:

Tabla Paises

Contiene información o relación de países en donde viven los clientes o empleados. La tabla **Paises** se encuentra en el esquema Venta

Columna	Tipo de datos	Nulos	Descripción
Idpais	char(3)	No NULL	Identificador de país. Clave primaria
NombrePais	Varchar(40)	No NULL	Nombre del país.

Tabla Categorias

Contiene información o relación de categorías en donde se encuentran registrados los productos. La tabla **Categorias** se encuentra en el esquema Compra.

Columna	Tipo de datos	Nulos	Descripción
IdCategoria	int	No NULL	Identificador de categoría. Clave primaria
NombreCategoria	Varchar(40)	No NULL	Nombre de la categoría.
Descripción	Text	Null	Descripción de la categoría

Tabla Clientes

Contiene información o relación de clientes que se encuentran registrados en la base de datos. La tabla **Clientes** se encuentra en el esquema Venta

Columna	Tipo de datos	Nulos	Descripción
IdCliente	Char(5)	No NULL	Identificador de cliente. Clave primaria
NomCliente	Varchar(40)	No NULL	Nombre del cliente.
DirCliente	Varchar(80)	No NULL	Dirección del cliente
Idpais	Char(3)	No NULL	Identificador de país. Clave externa de países.
fonoCliente	Varchar(15)	NULL	Teléfono del cliente

Tabla Proveedores

Contiene información o relación de los proveedores que se encuentran registrados en la base de datos. La tabla **Proveedores** se encuentra en el esquema Compra

Columna	Tipo de datos	Nulos	Descripción
IdProveedor	Int	No NULL	Identificador de proveedor. Clave primaria
nomProveedor	Varchar(80)	No NULL	Nombre del proveedor.
dirProveedor	Varchar(100)	No NULL	Dirección del proveedor.
nomContacto	Varchar(80)	No NULL	Nombre del contacto del proveedor.
cargoContacto	Varchar(50)	No NULL	Cargo del contacto del proveedor
idpais	Char(3)	No NULL	Identificador del país. Clave externa de países
fonoProveedor	Varchar(15)	No NULL	Teléfono del proveedor.
faxProveedor	Varchar(15)	No NULL	Fax del proveedor.

Tabla Productos

Contiene información o relación de los productos que ofrecen para la venta y que se encuentran registrados en la base de datos. La tabla **Productos** se encuentra en el esquema Compra.

Columna	Tipo de datos	Nulos	Descripción
IdProducto	Int	No NULL	Identificador de producto. Clave primaria
nomProducto	varchar(80)	No NULL	Nombre del producto.
idProveedor	Int	No NULL	Identificador del proveedor. Clave externa de proveedores
idCategoria	Int	No NULL	Identificador de la categoría. Clave externa de categorías.
cantxUnidad	varchar(50)	No NULL	Cantidad de productos por unidad almacenada
precioUnidad	decimal(10,2)	No NULL	Precio por unidad del producto
UniEnExistencia	smallint	No NULL	Unidades en existencia o stock del producto
UniEnPedido	smallint	No NULL	Unidades que se encuentran en pedido.

Tabla Cargos

Contiene información o relación de los cargos que se le asigna a cada empleado que se encuentran registrados en la base de datos. La tabla **Cargos** se encuentra en el esquema RRHH.

Columna	Tipo de datos	Nulos	Descripción
IdCargo	Int	No NULL	Identificador de cargo. Clave primaria
desCargo	varchar(30)	No NULL	Descripción del cargo

Tabla Distritos

Contiene información o relación de los distritos que se le asigna a cada empleado que se encuentran registrados en la base de datos. La tabla **Distritos** se encuentra en el esquema RRHH.

Columna	Tipo de datos	Nulos	Descripción
IdDistrito	Int	No NULL	Identificador de distrito. Clave primaria
nomDistrito	varchar(50)	No NULL	Nombre del distrito

Tabla Empleados

Contiene información o relación de los empleados que se encuentran registrados en la base de datos. La tabla **Empleados** se encuentra en el esquema RRHH.

Columna	Tipo de datos	Nulos	Descripción
IdEmpleado	Int	No NULL	Identificador del empleado. Clave primaria
nomEmpleado	varchar(50)	No NULL	Nombre del empleado
apeEmpleado	varchar(50)	No NULL	Apellido del empleado
fecNac	Datetime	No NULL	Fecha de Nacimiento
dirEmpleado	varchar(100)	No NULLL	Dirección del empleado
idDistrito	Int	No NULL	Identificador de distrito. Clave externa de distritos.
fonoEmpleado	varchar(15)	NULL	Teléfono del empleado
idcargo	Int	No NULL	Identificador de cargo, clave externa de cargos
fecContrata	Datetime	No NULL	Fecha de contratación
fotoEmpleado	Image	NULL	Foto del empleado

Tabla PedidosCabe

Contiene información o relación de la cabecera de los pedidos que se registran en el proceso de la venta y que se encuentran registrados en la base de datos. La tabla **PedidosCabe** se encuentra en el esquema Venta.

Columna	Tipo de datos	Nulos	Descripción
IdPedido	Int	No NULL	Identificador de la cabecera de pedido. Clave primaria
idcliente	varchar(5)	No NULL	Identificador de cliente. Clave externa de clientes
idEmpleado	Int	No NULL	Identificador del empleado. Clave externa de empleados
fechaPedido	Datetime	No NULL	Fecha de solicitud del pedido
fechaEntrega	Datetime	No NULL	Fecha de entrega del pedido
fechaEnvio	Datetime	No NULL	Fecha de envío del pedido
enviopedido	char(1)	No NULL	Indica si el pedido ha sido o no entregado
destinatario	varchar(60)	No NULL	Nombre del destinatario
dirdestinatario	varchar(100)	No NULL	Dirección del destinatario

Tabla PedidosDeta

Contiene información o relación del detalle de los productos solicitados en los pedidos de venta y que se encuentran registrados en la base de datos. La tabla **PedidosDeta** se encuentra en el esquema Compra.

Columna	Tipo de datos	Nulos	Descripción
IdPedido	Int	No NULL	Identificador de pedido. Clave externa de pedidoscabe
idProducto	Int	No NULL	Identificador del producto. Clave externa de producto
precioUnidad	Decimal(10,2)	No NULL	Precio del producto en el pedido
Cantidad	smallint	No NULL	Cantidad solicitada del producto
Descuento	Decimal(10,2)	No NULL	Cantidad de productos por unidad almacenada

1.1.2 Asignar nombres a los objetos de una Base de Datos

A menos que se especifique lo contrario, todas las referencias de Transact-SQL al nombre de un objeto de base de datos pueden ser un nombre de cuatro partes con el formato siguiente:

- *server_name*.[*database_name*].[*schema_name*].*object_name*
- *database_name*.[*schema_name*].*object_name*
- *schema_name*.*object_name*
- *object_name*

server_name: Especifica un nombre de servidor vinculado o un nombre de servidor remoto.

database_name: Especifica el nombre de una base de datos de SQL Server si el objeto reside en una instancia local de SQL Server. Cuando el objeto está en un servidor vinculado, *database_name* especifica un catálogo de OLE DB.

schema_name: Especifica el nombre del esquema que contiene el objeto si dicho objeto se encuentra en una base de datos de SQL Server. Si el objeto se encuentra en un servidor vinculado, *schema_name* especifica un nombre de esquema OLE DB.

object_name: Cuando se hace referencia a un objeto específico, no siempre hay que especificar el servidor, la base de datos y el esquema del SQL Server Database Engine (Motor de base de datos de SQL Server) para identificar el objeto. No obstante, si no se encuentra el objeto, se muestra un error.

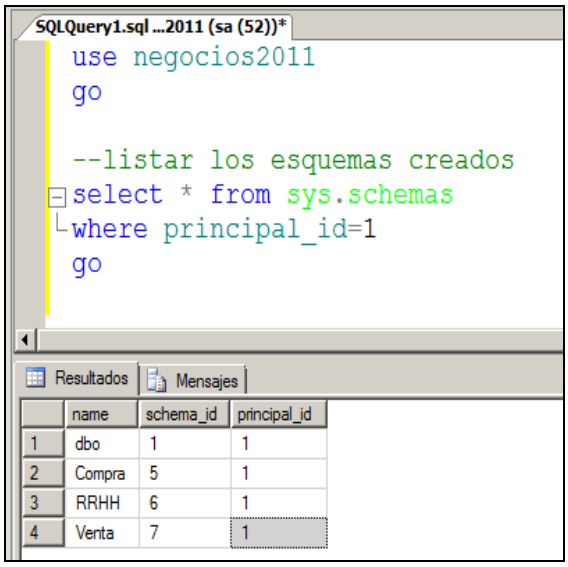
1.1.3 Manejo de Esquemas

Todos los objetos dentro de una base de datos, se crean dentro de un esquema. Los esquemas permiten agrupar objetos y ofrecer seguridad.

La definición de un esquema es simple, sólo se necesita identificar el comienzo de la definición con la instrucción **CREATE SCHEMA** y una cláusula adicional **AUTHORIZATION** y a continuación definir cada dominio, tabla, vista y demás en el esquema. Para crear los esquemas que se implementarán en la base de datos Negocios2011 autorizado por el propietario dbo:

```
USE NEGOCIOS2011
GO
```

```
-- CREAR LOS ESQUEMAS DE LA BASE DE DATOS
CREATE SCHEMA VENTA AUTHORIZATION DBO
GO

CREATE SCHEMA COMPRA AUTHORIZATION DBO
GO

CREATE SCHEMA RRHH AUTHORIZATION DBO
GO
```



Para listar los esquemas creados por el propietario de la base de datos (el database owner - dbo) se invoca a la tabla **sys.schemas**, tal como se muestra:

1.2 LENGUAJE DE MANIPULACION DE DATOS

1.2.1. Operadores

Un operador es un símbolo que especifica una acción que se realiza en una o más expresiones. A continuación, detallamos las categorías de operadores que utilizan SQL Server.

1.2.1.1. Operadores aritméticos

Son aquellos que realizan operaciones matemáticas entre dos expresiones numéricas.

Operador	Significado
+ (sumar)	Suma
- (restar)	Resta
* (multiplicar)	Multiplicación
/ (dividir)	División
% (Módulo)	Devuelve el resto entero de una división. Por ejemplo, 12 % 5 = 2 porque el resto de 12 dividido entre 5 es 2.

Los operadores de suma (+) y resta (-) son utilizados para realizar operaciones aritméticas sobre valores **datetime** y **smalldatetime**.

1.2.1.2. Operadores de Asignación

El operador (=) es sólo el operador de asignación del SQL Server. En el siguiente ejemplo, definimos la variable @num, asigne un valor a dicha variable.

```
DECLARE @NUM INT
SET @NUM=15
PRINT 'EL NUMERO INGRESADO ES:' + STR(@NUM)
```

El operador de asignación se utiliza para establecer encabezados de una columna. En el siguiente ejemplo, mostrar los encabezados de las columnas a la tabla Distritos.

1.2.1.3. Operadores de comparación

Los operadores de comparación permiten comprobar dos expresiones retornando un valor verdadero o falso, es decir, un dato **Boolean**. Se pueden utilizar en todas las expresiones excepto en las de los tipos de datos **text**, **ntext** o **image**. En la siguiente tabla, se presentan los operadores de comparación Transact-SQL.

Operador de Comparación	Significado
=	Igual a
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que

<>	No es igual a
!=	No es igual a (no es del estándar ISO)
!<	No es menor que (no es del estándar ISO)
!>	No es mayor que (no es del estándar ISO)

1.2.1.4. Operadores lógicos

Los operadores lógicos comprueban la veracidad de alguna condición. Éstos, como los operadores de comparación, devuelven el tipo de datos Boolean con el valor TRUE, FALSE o UNKNOWN.

Operador	Significado
ALL	TRUE si el conjunto completo de comparaciones es TRUE.
AND	TRUE si ambas expresiones booleanas son TRUE.
ANY	TRUE si cualquier miembro del conjunto de comparaciones es TRUE.
BETWEEN	TRUE si el operando está dentro de un intervalo.
EXISTS	TRUE si una subconsulta contiene cualquiera de las filas.
IN	TRUE si el operando es igual a uno de la lista de expresiones.
LIKE	TRUE si el operando coincide con un patrón.
NOT	Invierte el valor de cualquier otro operador booleano.
OR	TRUE si cualquiera de las dos expresiones booleanas es TRUE.
SOME	TRUE si alguna de las comparaciones de un conjunto es TRUE.

1.2.1.5. Operador BETWEEN

Especifica un intervalo que se va a evaluar, retorna un valor *boolean*; retorna TRUE si el valor de la **expresión a evaluar** es mayor o igual que el valor de **inicio expresión** y menor o igual que el valor de **fin expresión**.

NOT BETWEEN devuelve TRUE si el valor de **expresión a evaluar** es menor que el valor de **inicio expresión** y mayor que el valor de **fin expresión**.

Sintaxis:

```
EXPRESIÓN_A_EVALUAR [NOT] BETWEEN INICIO_EXPRESIÓN AND
FIN_EXPRESIÓN
```

Ejemplo: Mostrar todos los productos donde el valor del precioUnidad se encuentre entre 27 a 30

```
USE NEGOCIOS2011
GO

SELECT     P.NOMPRODUCTO 'PRODUCTO',
           C.NOMCATEGORIA 'CATEGORIA'
FROM COMPRA.PRODUCTOS P
JOIN COMPRA.CATEGORIAS C ON P.IDCATEGORIA = C.IDCATEGORIA
WHERE P.PRECIOUNIDAD BETWEEN 27 AND 30
ORDER BY P.NOMPRODUCTO
GO
```

1.2.1.6. Operador LIKE

Determina si una cadena de caracteres específica coincide con un patrón determinado. Un patrón puede contener caracteres normales y caracteres comodín. Durante la operación de búsqueda de coincidencias de patrón, los caracteres normales deben coincidir exactamente con los caracteres especificados en la cadena de caracteres. Sin embargo, los caracteres comodín pueden coincidir con fragmentos arbitrarios de la cadena. La utilización de caracteres comodín hace que el operador LIKE sea más flexible que los operadores de comparación de cadenas = y !=.

Sintaxis

```
MATCH_EXPRESSION [NOT] LIKE PATTERN [ESCAPE ESCAPE_CHARACTER]
```

Argumentos:

match_expression: Es cualquier expresión válida de tipo de datos de caracteres.

Pattern: Es la cadena de caracteres específica que se busca en *match_expression*; puede incluir los siguientes caracteres comodín válidos. *pattern* puede tener 8.000 bytes como máximo.

Carácter comodín	Descripción	Ejemplo
%	Cualquier cadena de cero o más caracteres.	WHERE title LIKE '%computer%' busca todos los títulos de libros que contengan la palabra 'computer' en el título.
_ (carácter de subrayado)	Cualquier carácter.	WHERE au_fname LIKE '_ean' busca todos los nombres de cuatro letras que terminen en ean (Dean, Sean, etc.)
[ ]	Cualquier carácter del intervalo ([a-f]) o conjunto ([abcdef]) que se ha especificado.	WHERE au_lname LIKE '[C-P]arsen' busca apellidos de autores que terminen en arsen y empiecen por cualquier carácter individual entre C y P, como Carsen, Larsen, Karsen, etc.
[^]	Cualquier carácter que no se encuentre en el intervalo ([^a-f]) o conjunto ([^abcdef]) que se ha especificado.	WHERE au_lname LIKE 'de[^l]%' busca todos los apellidos de autores que empiecen por de y en los que la siguiente letra no sea l.

escape_character: Es un carácter que se coloca delante de un carácter comodín para indicar que el comodín no debe interpretarse como un comodín, sino como un carácter normal. *escape_character* es una expresión de caracteres que no tiene ningún valor predeterminado y se debe evaluar como un único carácter.

Ejercicio:

```
USE NEGOCIOS2011
GO

-- RETORNA LOS REGISTROS DE EMPLEADOS DONDE SU APELLIDO TERMINE
EN KING
SELECT * FROM RRHH.EMPLEADOS
WHERE APEEMPLEADO LIKE '%KING'
GO

-- RETORNA LOS REGISTROS DE EMPLEADOS DONDE SU APELLIDO INICIE
CON KING
SELECT * FROM RRHH.EMPLEADOS
WHERE APEEMPLEADO LIKE 'KING%'
GO
```

```
-- RETORNA LOS REGISTROS DE EMPLEADOS DONDE SU APELLIDO
CONTENGA LA EXPRESION KING
SELECT * FROM  RRHH.EMPLEADOS
WHERE APEEMPLEADO LIKE '%KING%'
GO
```

1.2.2. Funciones para el manejo de datos

1.2.2.1. Funciones para el manejo de fechas

Función	Descripción

DATEADD	Devuelve un valor *date* con el intervalo *number* especificado, agregado a un valor *datepart* especificado de ese valor *date*.
	
	DATEADD (datepart , number , date)
	
	`DECLARE @FECHA DATE = '1-8-2011'`
	`SELECT 'YEAR' 'PERIODO ', DATEADD(YEAR,1,@FECHA) 'NUEVA FECHA'`
	`UNION ALL`
	`SELECT 'QUARTER',DATEADD(QUARTER,1,@FECHA)`
	`UNION ALL`
	`SELECT 'MONTH',DATEADD(MONTH,1,@FECHA)`
	`UNION ALL`
	`SELECT 'DAY',DATEADD(DAY,1,@FECHA)`
	`UNION ALL`
	`SELECT 'WEEK',DATEADD(WEEK,1,@FECHA)`
	`GO`

DATEDIFF	Devuelve el número de límites *datepart* de fecha y hora entre dos fechas especificadas.
	
	DATEDIFF (datepart , startdate , enddate)
	
	`SET DATEFORMAT DMY`
	`DECLARE @FECHAINICIAL DATE = '01-08-2011';`
	`DECLARE @FECHAFINAL DATE = '01-09-2011';`
	`SELECT DATEDIFF(DAY, @FECHAINICIAL,@FECHAFINAL) AS 'DURACION'`

DATENAME	Devuelve una cadena de caracteres que representa el *datepart* especificado de la fecha especificada. **DATENAME (*datepart* , *date*)** `SELECT DATENAME(MONTH, GETDATE()) AS 'MES';`
DATEPART	Devuelve un entero que representa el *datepart* especificado del *date* especificado. **DATEPART (*datepart* , *date*)** `SELECT DATEPART(MONTH, GETDATE()) AS 'MES';`
DAY	Devuelve un entero que representa la parte del día **datepart** de la fecha especificada. `SELECT DAY('01/9/2011') AS 'DÍA DEL MES';`
GETDATE	Devuelve la fecha del sistema `SELECT GETDATE() 'FECHA DEL SISTEMA';`
MONTH	Devuelve un entero que representa el mes de *date* especificado. MONTH devuelve el mismo valor que DATEPART (**month**, *date*). `SELECT MONTH(GETDATE()) AS 'MES DE LA FECHA DE SISTEMA';`
YEAR	Devuelve un entero que representa el año de *date* especificado. YEAR devuelve el mismo valor que DATEPART (**year**, *date*). `SELECT YEAR(GETDATE()) AS 'AÑO DE LA FECHA DE SISTEMA';`

1.2.2.2. Funciones para el manejo de cadenas

Función	Descripción
LEFT	Devuelve la parte izquierda de una cadena de caracteres con el número de caracteres especificado. **LEFT (character_expression , integer_expression)**

LEN	Devuelve el número de caracteres de la expresión de cadena especificad, excluidos los espacios en blanco finales. **LEN (string_expression)**
LOWER	Devuelve una expresión de caracteres después de convertir en minúsculas los datos de caracteres en mayúsculas. **LOWER (character_expression)**
LTRIM	Devuelve una expresión de caracteres tras quitar todos los espacios iniciales en blanco. **LTRIM (character_expression)**
RTRIM	Devuelve una cadena de caracteres después de truncar todos los espacios en blanco finales. **RTRIM (character_expression)**
SUBSTRING	Devuelve parte de una expresión de caracteres, binaria, de texto o de imagen. Para obtener más información acerca de los tipos de datos válidos de SQL Server que se pueden usar con esta función. **SUBSTRING (value_expression, start_expression, length_expression)**
UPPER	Devuelve una expresión de caracteres con datos de caracteres en minúsculas convertidos a mayúsculas. **UPPER (character_expression)**

Ejercicio

```
-- MANEJO DE CADENAS: RETORNA LA EXPRESION BASE CONVERTIDA EN
MAYÚSCULAS
DECLARE @CADENA VARCHAR(30)
SELECT @CADENA = ' BASE DE DATOS AVANZADO ';
SELECT LEFT(UPPER(LTRIM(@CADENA)),4) AS 'CADENA RESULTANTE'
GO
```

1.2.2.3. Funciones de conversión

Convierte una expresión de un tipo de datos en otro tipo de dato definido en SQL Server 2008.

Función	Descripción
CAST	Convierte una expresión a un tipo de datos `CAST (expresión AS tipo_dato[(longitud)])`
CONVERT	Convierte una expresión a un tipo de datos indicando un estilo. `CONVERT (tipo_dato [(longitud)], expresión [, estilo])`

Ejemplo

```
USE NEGOCIOS2011
GO

SELECT DISTINCT CAST(P.NOMPRODUCTO AS CHAR(15)) AS NOMBRE,
CONVERT(DECIMAL(10,2),P.PRECIOUNIDAD) AS 'PRECIO UNITARIO'
FROM COMPRA.PRODUCTOS
WHERE P.NOMPRODUCTO LIKE 'PAN%';
GO
```

1.2.3. Comandos de LMD (Lenguaje de Manipulación de Datos)

1.2.3.1. Insertar registros: INSERT

Agrega una o varias filas nuevas a una tabla o una vista en SQL Server 2008.

Sintaxis**:**

```
INSERT
{       [TOP (expresión) [ PERCENT ] ] [ INTO ]  { <OBJETO> }
    {
        { VALUES ( { DEFAULT | NULL | expression } [ ,...n ] )
[ ,...n ]
          | table_derivada
      | sentencia_ejecutar
      | <table_origen>
      | DEFAULT VALUES
        }
    }
}
```

El formato básico de la sentencia es:

INSERT INTO tabla [(columna1, columna2, columnan)] **VALUES** (expr1, expr2, exprn)

- **Tabla** es el nombre de la tabla donde se desea ingresar los nuevos datos.
- **Columna** es una lista opcional de nombres de campo en los que se insertarán valores en el mismo número y orden que se especificarán en la cláusula VALUES. Si no se especifica la lista de campos, los valores de **expr** en la cláusula VALUES deben ser tantos como campos tenga la tabla y en el mismo orden que se definieron al crear la tabla.
- **Expr** es una lista de expresiones o valores constantes, separados por comas, para dar valor a los distintos campos del registro que se añadirá a la tabla. Las cadenas de caracteres deberán estar encerradas entre apóstrofes.

1.2.3.1.1. **Insertar un único registro**

A. Especificando todos los campos a ingresar.

Cada sentencia **INSERT** añade un único registro a la tabla.
En el ejemplo, se han especificado todos los campos con sus respectivos valores. Si no se ingresara valores a un campo, este se cargará con el valor **DEFAULT** o ***NULL*** (siempre y cuando haya sido especificado en la estructura de la tabla). Un valor nulo –*NULL*- no significa blancos o ceros, sino que el campo nunca ha tenido un valor.

```
USE NEGOCIOS2011
GO

INSERT INTO VENTA.CLIENTES(IDCLIENTE,NOMCLIENTE, DIRCLIENTE,
IDPAIS, FONOCLIENTE)
VALUES ('DRATR', 'DARIO TRAGODARA', 'CALLE LUIS MIRO 123',
'003', '3245566');
GO

SELECT * FROM VENTA.CLIENTES
GO
```

B. Especificando únicamente los valores de los campos.

Si no se especifica la lista de campos, los valores en la cláusula VALUES deben ser tantos como campos tenga la tabla y en el mismo orden que se definieron al crear la tabla. Si se va a ingresar parcialmente los valores en una tabla, se debe especificar el nombre de los campos a ingresar, como en el ejemplo **A**.

```
USE NEGOCIOS2011
GO

INSERT INTO VENTA.CLIENTES
VALUES ('DRAPR', 'DARIO PRADO', 'CALLE 32', '001', '3245566');
GO

SELECT * FROM VENTA.CLIENTES
GO
```

1.2.3.1.2. Insertar varias filas de datos

En el siguiente ejemplo, se usa el constructor de valores de tabla para insertar tres filas en la tabla Venta.Paises en una instrucción INSERT. Dado que los valores para todas las columnas se suministran e incluyen en el mismo orden que las columnas de la tabla, no es necesario especificar los nombres de columna en la lista de columnas.

```
USE NEGOCIOS2011
GO

INSERT INTO VENTA.PAISES
VALUES ('095', 'NORUEGA'), ('096', 'ISLANDIA'), ('097',
'GRECIA');
GO

SELECT * FROM VENTA.PAISES P
WHERE P.IDPAIS  IN ('095','096','097')
GO
```

A. Insertar Múltiples Registros

Utilizando el comando SELECT, podemos agregar múltiples registros. Veamos un ejemplo:

```
USE NEGOCIOS2011
GO

CREATE TABLE RRHH.EMPLEADOS2011(
    IDEMPLEADO INT NOT NULL,
    NOMEMPLEADO VARCHAR(50) NOT NULL,
    APEEMPLEADO VARCHAR(50) NOT NULL,
    FONOEMPLEADO VARCHAR(15) NULL,
    DIREMPLEADO VARCHAR(100) NOT NULL,
    IDDISTRITO INT NOT NULL
)
GO

INSERT INTO RRHH.EMPLEADOS2011
SELECT A.IDEMPLEADO, A.NOMEMPLEADO, A.APEEMPLEADO,
A.FONOEMPLEADO,
           A.DIREMPLEADO, A.IDDISTRITO
FROM RRHH.EMPLEADOS AS A
WHERE YEAR(A.FECCONTRATA) = '2011
GO
```

```
SELECT * FROM RRHH.EMPLEADOS2011
GO
```

B. Insertar datos en una variable de tabla

En el siguiente ejemplo, se especifica una variable de tabla como el objeto de destino.

```
USE NEGOCIOS2011;
GO

-- CREA UNA VARIABLE TIPO TABLA
DECLARE @PRODUCTO TABLE(
    PRODUCTOID INT NOT NULL,
    PRODUCTONOMBRE VARCHAR(100) NOT NULL,
    PRODUCTOPRE AS DECIMAL,
    PRODUCTOCAN INT);
GO

-- INSERTA VALORES DENTRO DE LA VARIABLE TIPO TABLA
INSERT INTO @PRODUCTO (PRODUCTOID, PRODUCTONOMBRE, PRODUCTOPRE,
PRODUCTOCAN)
SELECT IDPRODUCTO, NOMPRODUCTO, PRECIOUNIDAD,
UNIDADESENEXISTENCIA
FROM COMPRA.PRODUCTOS
WHERE PRECIOUNIDAD > 100;

--VER EL CONJUNTO DE VALORES DE LA VARIABLE TIPO TABLA
SELECT * FROM @ PRODUCTO;
GO
```

C. Insertar datos en una tabla con columnas que tienen valores predeterminados

```
USE NEGOCIOS2011;
GO

CREATE TABLE DBO.PRUEBA
(
    COLUMNA_1 AS 'COLUMNA CALCULADA ' + COLUMNA_2,
    COLUMNA_2 VARCHAR(30) DEFAULT ('COLUMNA POR DEFECTO'),
    COLUMNA_3 ROWVERSION,
    COLUMNA_4 VARCHAR(40) NULL
)
GO


INSERT INTO DBO.PRUEBA (COLUMN_4) VALUES ('VALOR');
INSERT INTO DBO.PRUEBA (COLUMN_2, COLUMN_4) VALUES ('VALOR',
'VAL');
INSERT INTO DBO.PRUEBA (COLUMN_2) VALUES ('VALOR');
INSERT INTO PRUEBA DEFAULT VALUES;
GO

SELECT COLUMNA_1, COLUMAN_2, COLUMNA_3, COLUMNA_4
FROM DBO.PRUEBA;
GO
```

1.2.3.2. Actualización de datos: UPDATE

La sentencia *UPDATE* se utiliza para cambiar el contenido de los registros de una o varias columnas de una tabla de la base de datos. Su formato es:

```
UPDATE Nombre_tabla
SET nombre_columna1 = expr1, nombre_columna2 = expr2,…...
[WHERE {condición}]
```

- **Nombre_tabla** nombre de la tabla donde se cambiará los datos.
- **Nombre_columna** columna cuyo valor se desea cambiar. En una misma sentencia *UPDATE* pueden actualizarse varios campos de cada registro.
- **Expr** es el nuevo valor que se desea asignar al campo. La expresión puede ser un valor constante o una subconsulta. Las cadenas de caracteres deberán estar encerradas entre comillas. Las subconsultas entre paréntesis.

La cláusula *WHERE* sigue el mismo formato que la vista en la sentencia *SELECT* y determina qué registros se modificarán.

1.2.3.2.1. **Actualizar varias columnas**

En el siguiente ejemplo, se actualizan los valores de las columnas **precioUnidad** y **UnidadesEnExistencia** para todas las filas de la tabla Productos.

```
USE NEGOCIOS2011;
GO

UPDATE COMPRA.PRODUCTOS
SET PRECIOUNIDAD = 6000, UNIDADESENEXISTENCIA *= 1.50
GO
```

1.2.3.2.2. **Limitar las filas que se actualizan usando la cláusula WHERE**

En el ejemplo siguiente, actualice el valor de la columna precioUnidad de la tabla Compra.Productos incrementando su valor en un 25% más, para todas las filas cuyo nombre del producto inicie con “A” y su stock o unidadesenExistencia sea mayor a 100.

```
USE NEGOCIOS2011;
GO

UPDATE COMPRA.PRODUCTOS
SET PRECIOUNIDAD *= 1.25
WHERE NOMPRODUCTO LIKE 'A%' AND UNIDADESENEXISTENCIA > 100;
GO
```

1.2.3.2.3. Usar la instrucción UPDATE con información de otra tabla

En este ejemplo, se modifica la columna ventaEmp de la tabla SalesEmpleado para reflejar las ventas registradas en la tabla Pedidos.

```
USE NEGOCIOS2011;
GO

UPDATE VENTA.SALESEMPLEADO
SET VENTAEMP = VENTAEMP + (SELECT SUM(PRECIOUNIDAD*CANTIDAD)
      FROM VENTA.PEDIDOSCABE PE JOIN VENTA.PEDIDOSDETA AS PD
      ON PE.IDPEDIDO= PD.IDPEDIDO)
GO
```

1.2.3.3. Eliminación de datos: DELETE

La sentencia *DELETE* se utiliza para eliminar uno o varios registros de una misma tabla. En una instrucción *DELETE* con múltiples tablas, debe incluir el nombre de tabla (Tabla.*). Si especifica más de una tabla para eliminar registros, todas deben tener una relación de muchos a uno. Si desea eliminar todos los registros de una tabla, eliminar la propia tabla es más eficiente que ejecutar una consulta de borrado.

Las operaciones de eliminación en cascada en una consulta únicamente eliminan desde varios lados de una relación. Por ejemplo, en la base de datos NEGOCIOS2011, la relación entre las tablas Clientes y PedidosCabe, la tabla PedidosCabe es la parte de muchos, por lo que las operaciones en cascada sólo afectarán a la tabla PedidosCabe. Una consulta de borrado elimina los registros completos, no únicamente los datos en campos específicos. Si desea eliminar valores en un campo especificado, crea una consulta de actualización que cambie los valores a *Null*.

El formato de la sentencia es:

```
DELETE FROM Nombre_Tabla
[WHERE { condición }]
```

- **Nombre_Tabla** es el nombre de la tabla donde se desea borrar los datos.
- La cláusula ***WHERE*** sigue el mismo formato que la vista en la sentencia *SELECT* y determina qué registros se borrarán.

1.2.3.3.1. Eliminar registros

En el siguiente ejemplo, elimine los registros de la tabla PedidosCabe. Cada sentencia *DELETE* borra los registros que cumplen la condición impuesta o todos si no se indica cláusula *WHERE*

```
USE NEGOCIOS2011;
GO

DELETE FROM VENTA.PEDIDOSCABE
GO
```

1.2.3.3.2. Eliminar las filas usando la cláusula WHERE

En el ejemplo siguiente, elimine los registros de la tabla PedidosDeta de todos aquellos pedidos cuya antigüedad sea mayor a 10 años.

```
USE NEGOCIOS2011;
GO

DELETE VENTA.PEDIDOSDETA
FROM  VENTA.PEDIDOSCABE PE JOIN VENTA.PEDIDOSDETA PD
ON PE.IDPEDIDO=PD.IDPEDIDO
WHERE DATEDIFF(YY, GETDATE(), FECHAPEDIDO) > 10;
GO
```

1.2.3.4. Selección de datos : SELECT

Recupera las filas de la base de datos y habilita la selección de una o varias filas o columnas de una o varias tablas en SQL Server 2008.
La sintaxis completa de la instrucción SELECT es compleja, aunque las cláusulas principales se pueden resumir del modo siguiente:

Sintaxis:

```
<SELECT statement> ::=
    [WITH <common_table_expression> [,...n]]
    <query_expression>
    [ ORDER BY { order_by_expression | column_position [ ASC |
DESC ] }
  [ ,...n ] ]
    [ COMPUTE
  { { AVG | COUNT | MAX | MIN | SUM } (expression )} [ ,...n ]
  [ BY expression [ ,...n ] ]
    ]
    [ <FOR Clause>]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
<query_expression> ::=
    { <query_specification> | ( <query_expression> ) }
    [  { UNION [ ALL ] | EXCEPT | INTERSECT }
        <query_specification> | ( <query_expression> ) [...n ] ]
<query_specification> ::=
SELECT [ ALL | DISTINCT ]
    [TOP (expression) [PERCENT] [ WITH TIES ] ]
    < select_list >
    [ INTO new_table ]
    [ FROM { <table_source> } [ ,...n ] ]
    [ WHERE <search_condition> ]
    [ <GROUP BY> ]
    [ HAVING < search_condition > ]
```

Para nuestro curso usaremos la siguiente sintaxis:

```
SELECT [ALL|DISTINCT] [TOP (expresión) [PERCENT] [WITH TIES] ]
    < lista de selección >
      [INTO nombre de la nueva tabla]
FROM <nombre de tabla>
WHERE <condición>
GROUP BY <nombre de campos>
HAVING <condición> [AND | OR <condición>]
ORDER BY
```

1.2.3.4.1. **Orden de procesamiento lógico de la instrucción SELECT**

Los pasos siguientes muestran el orden de procesamiento lógico, u orden de enlace, para una instrucción SELECT. Este orden determina el momento en que los objetos definidos en un paso están disponibles para las cláusulas de los pasos subsiguientes. Por ejemplo, si el procesador de consultas se puede enlazar (obtener acceso) a las tablas o vistas definidas en la cláusula FROM, estos objetos y sus columnas quedan disponibles para todos los pasos subsiguientes. A la inversa, dado que la cláusula SELECT es el paso 8, las cláusulas precedentes no pueden hacer referencia a los alias de columna o las columnas derivadas definidos en esa cláusula. Sin embargo, las cláusulas subsiguientes, como la cláusula ORDER BY, sí pueden hacer referencia a ellos. Observe que la ejecución física real de la instrucción está determinada por el procesador de consultas y el orden de esta lista puede variar.

1. FROM
2. ON
3. JOIN
4. WHERE
5. GROUP BY
6. WITH CUBE o WITH ROLLUP
7. HAVING
8. SELECT
9. DISTINCT
10. ORDER BY
11. TOP

Ejemplo: Recupera las filas de la tabla Productos cuyo precioUnidad sea mayor a 50

```
USE NEGOCIOS2011;
GO

BEGIN
DECLARE @MYPRODUCTO INT
SET @MYPRODUCTO = 750
--  EVALUAR SI LA VARIABLE @MYPRODUCTO ES DIFERENTE DE 0
IF (@MYPRODUCTO <> 0)
```

```
   SELECT  IDPRODUCTO 'CODIGO',
           NOMPRODUCTO  'PRODUCTO',
           PRECIOUNIDAD 'PRECIO'
   FROM COMPRA.PRODUCTOS
   WHERE IDPRODUCTO = @MYPRODUCTO;
END
GO
```

1.2.3.4.2. **Crear una tabla a partir de una consulta**

Utilice la siguiente sintaxis para la creación de una tabla con datos a partir de una consulta:

```
SELECT <CAMPOS> INTO TABLA
FROM TABLA_EXISTENTE
WHERE <CONDICION>
```

Por ejemplo: Recuperar los registros de empleados cuyo cargo sea Supervisor de Ventas y almacenarlos en la tabla EmpleadosBAK

```
USE NEGOCIOS2011
GO

SELECT IDEMPLEADO,
       APEEMPLEADO,
       NOMEMPLEADO
INTO DBO.EMPLEADOBAK
FROM RRHH.EMPLEADOS
WHERE IDCARGO = (SELECT C.IDCARGO
                 FROM RRHH.CARGOS C
                 WHERE DESCARGO = 'SUPERVISOR DE VENTAS')
GO

SELECT * FROM DBO. EMPLEADOBAK
GO
```

1.2.4. INSTRUCCION MERGE

La instrucción MERGE, nos permite realizar múltiples acciones sobre una tabla tomando uno o varios criterios de comparación; es decir, realiza operaciones de inserción, actualización o eliminación en una tabla de destino según los resultados de una combinación con una tabla de origen. Por ejemplo, puede sincronizar dos tablas insertando, actualizando o eliminando las filas de una tabla según las diferencias que se encuentren en la otra.

La instrucción MERGE nos sirve básicamente para dos cosas:

1 Sincronizar los datos de 2 tablas. Supongamos que tenemos 2 bases distintas (Producción y Desarrollo por ejemplo) y queremos sincronizar los datos de una tabla para que queden exactamente iguales. Lo que antes hubiese implicado algunas sentencias mezcladas con INNER JOIN y NOT EXISTS, ahora es posible resumirlo en una operación atómica mucho más sencilla y eficiente.

2 La otra razón por la cual podríamos usar MERGE, es cuando tenemos nuevos datos que queremos almacenar en una tabla y no sabemos si la primary key de la tabla ya existe o no, por lo tanto, no sabemos si hacer un UPDATE o un INSERT en la tabla.

Sintaxis:

```
MERGE [INTO] <target table>
USING <source table>
ON <join/merge predicate>
WHEN [TARGET] NOT MATCHED <statement to runt>
```

Donde:

<target table>: Es la tabla de destino de las operaciones de inserción, actualización o eliminación que las cláusulas WHEN de la instrucción MERGE especifican.

<source table>: Especifica el origen de datos que se hace coincidir con las filas de datos en *target_table*. El resultado de esta coincidencia dicta las acciones que tomarán las cláusulas WHEN de la instrucción MERGE.

<join/merge predicate>: Especifica las condiciones en las que **table_source** se combina con **target_table** para determinar dónde coinciden.

<statement to run when match found in target>: Especifica que todas las filas de *target_table* que coinciden con las filas que devuelve <table_source> ON <merge_search_condition> y que satisfacen alguna condición de búsqueda adicional se actualizan o eliminan según la cláusula <merge_matched>.

La instrucción MERGE puede tener a lo sumo dos cláusulas WHEN MATCHED. Si se especifican dos cláusulas, la primera debe ir acompañada de una cláusula AND <search_condition>. Si hay dos cláusulas WHEN MATCHED, una debe especificar una acción UPDATE y la otra una acción DELETE. Puede actualizar la misma fila más de una vez, ni actualizar o eliminar la misma fila.

Ejemplo: **Usar MERGE para realizar operaciones INSERT y UPDATE en una tabla en una sola instrucción**. Implemente un escenario para actualizar o insertar un registro a la tabla países: Si existe el código del país, actualice su nombre; sino inserte el registro a la tabla

```
USE NEGOCIOS2011
GO

DECLARE @PAIS VARCHAR(50), @ID CHAR(3)
SET @PAIS='NIGERIA'
SET @ID='99'

MERGE VENTAS.PAISES AS TARGET
USING (SELECT @ID, @PAIS)
AS SOURCE (IDPAIS, NOMBREPAIS)
ON (TARGET.IDPAIS = SOURCE.IDPAIS)
WHEN MATCHED THEN
      UPDATE SET NOMBREPAIS = SOURCE.NOMBREPAIS
WHEN NOT MATCHED THEN
      INSERT VALUES(SOURCE.IDPAIS, SOURCE.NOMBREPAIS);
GO
```

Ejemplo: **Usar MERGE para realizar operaciones DELETE y UPDATE en una tabla en una sola instrucción**. Implemente un escenario para actualizar o eliminar un registro a la tabla productos: Si existe el código del producto y las unidadesEnExistencia es menor o igual a cero, elimine el registro; sino actualice el nombre del producto

```
USE NEGOCIOS2011
GO

DECLARE @PRODUCTO VARCHAR(50), @ID INT
SET @PRODUCTO = 'VINO'
SET @ID = 22

MERGE COMPRAS.PRODUCTOS AS TARGET
USING (SELECT @ID, @PRODUCTO)
AS SOURCE (IDPRODUCTO, NOMPRODUCTO)
ON (TARGET.IDPRODUCTO = SOURCE.IDPRODUCTO)
WHEN MATCHED AND TARGET.UNIDADESENEXISTENCIA<=0 THEN
     DELETE
WHEN MATCHED THEN
     UPDATE  SET NOMPRODUCTO = SOURCE.NOMPRODUCTO;
GO
```

Ejemplo: **Usar MERGE para realizar operaciones INSERT, DELETE y UPDATE en una tabla en una sola instrucción**. Implemente un escenario para insertar, actualizar o eliminar un registro a la tabla clientesBAK: Si existe el código del cliente, si el nombre del cliente y su dirección no coincide, actualice sus datos; sino existe el código Inserte el registro; y si no coincide en el origen elimine el Registro

```
USE NEGOCIOS2011
GO

MERGE VENTAS.CLIENTESBAK AS TARGET
USING VENTAS.CLIENTES AS SOURCE
ON (TARGET.IDCLIENTE = SOURCE.IDCLIENTE)
WHEN MATCHED AND TARGET.NOMBRECLIENTE <> SOURCE .NOMBRECLIENTE THEN
     UPDATE SET TARGET.NOMBRECLIENTE = SOURCE .NOMBRECLIENTE,
```

```
                    TARGET.DIRCLIENTE = SOURCE .DIRCLIENTE
WHEN NOT MATCHED THEN
      INSERT  VALUE(SOURCE.NOMBRECLIENTE, SOURCE .DIRCLIENTE)
WHEN NOT MATCHED BY SOURCE THEN
      DELETE
GO
```

1.3. RECUPERACIÓN AVANZADA DE CONSULTA DE DATOS

Conocidos los fundamentos básicos del comando SELECT, y está familiarizado con varias de sus cláusulas, a continuación vamos a aprender técnicas de consultas más avanzadas. Una de estas técnicas es la de combinar el contenidos de una o más tablas para producir un conjunto de resultados que incorpore filas y columnas de cada tabla. Otra técnica es la de agrupar los datos para obtener, desde un conjunto de filas, datos agrupados. Se pueden usar los elementos de Transact-SQL tales con CUBE y ROLLUP para resumir datos.

1.3.1. COMBINACION DE TABLAS: JOIN

La sentencia JOIN en el lenguaje de consulta, permite combinar registros de dos o más tablas en una base de datos relacional. La sentencia JOIN se pueden especificar en las cláusulas *FROM* o *WHERE*, aunque se recomienda que se especifiquen en la cláusula *FROM*.

Las combinaciones se pueden clasificar en:

1.3.1.1. COMBINACION INTERNA.

Con esta operación, se calcula el producto cruzado de los registros de dos tablas, pero solo permanecen aquellos registros en la tabla combinada que satisfacen las condiciones que se especifican. La cláusula **INNER JOIN** permite la combinación de los registros de las tablas, comparando los valores de la columna específica en ambas tablas. Cuando no existe esta correspondencia, el registro no se muestra
Esta consulta de *Transact SQL* es un ejemplo de una combinación interna:

```
USE NEGOCIOS2011
GO

SELECT C.IDCLIENTE, C.NOMCLIENTE, C.DIRCLIENTE, P.NOMBREPAIS
FROM VENTAS.CLIENTES C INNER JOIN VENTAS.PAISES P
ON C.IDPAIS = P.IDPAIS
GO
```

Esta combinación interna se conoce como una combinación equivalente. Devuelve todas las columnas de ambas tablas y sólo devuelve las filas en las que haya un valor igual en la columna de la combinación. Es equivalente a la siguiente consulta:

```
USE NEGOCIOS2011
GO

SELECT C.IDCLIENTE, C.NOMCLIENTE, C.DIRCLIENTE, P.NOMBREPAIS
FROM VENTAS.CLIENTES C, VENTAS.PAISES P
WHERE  C.IDPAIS = P.IDPAIS
GO
```

1.3.1.2. COMBINACIONES EXTERNAS

Mediante esta operación no se requiere que cada registro en las tablas a tratar tenga un registro equivalente en la otra tabla. El registro es mantenido en la tabla combinada si no existe otro registro que le corresponda. Este tipo de operación se subdivide dependiendo de la tabla a la cual se le admitirán los registros que no tienen correspondencia, ya sean de tabla izquierda, de tabla derecha o combinación completa.

SQL Server 2008 utiliza las siguientes palabras clave para las combinaciones externas especificadas en una cláusula *FROM*:

- *LEFT OUTER JOIN* o ***LEFT JOIN***
- *RIGHT OUTER JOIN* o ***RIGHT JOIN***
- *FULL OUTER JOIN* o ***FULL JOIN***

A. LEFT JOIN o LEFT OUTER JOIN

La sentencia LEFT JOIN retorna la pareja de todos los valores de la izquierda con los valores de la tabla de la derecha correspondientes, o retorna un valor nulo NULL en caso de no correspondencia.

El operador de combinación *LEFT JOIN*, indica que todas las filas de la primera tabla se deben incluir en los resultados, con independencia si hay datos coincidentes en la segunda tabla.

Ejemplo: Mostrar los registros de los clientes que han solicitado pedidos y aquellos clientes que aun no han registrado pedidos

```
USE NEGOCIOS2011
GO

SELECT C.*, P.IDPEDIDO
FROM VENTAS.CLIENTES C INNER JOIN VENTAS.PEDIDOSCABE P
ON C.IDCLIENTE = P.IDCLIENTE
GO
```

B. RIGHT JOIN o RIGHT OUTER JOIN

Una combinación externa derecha es el inverso de una combinación externa izquierda. Se devuelven todas las filas de la tabla de la derecha. Cada vez que una fila de la tabla de la derecha no tenga correspondencia en la tabla de la izquierda, se devuelven valores *NULL* para la tabla de la izquierda.

El operador de combinación *RIGHT JOIN*, indica que todas las filas de la segunda tabla se deben incluir en los resultados, con independencia si hay datos coincidentes en la primera tabla.

Ejemplo: Mostrar los pedidos registrados por los productos, incluya los productos que aun no se ha registrado en algún pedido.

```
USE NEGOCIOS2011
GO
```

```
SELECT PD.*, PO.NOMPRODUCTO
FROM COMPRAS.PRODUCTOS PO RIGHT JOIN VENTAS.PEDIDOSDETA PD
ON PD.IDPRODUCTO = PO.IDPRODUCTO
GO
```

C. FULL JOIN o FULL OUTER JOIN

Una combinación externa completa devuelve todas las filas de las tablas de la izquierda y la derecha. Cada vez que una fila no tenga coincidencia en la otra tabla, las columnas de la lista de selección de la otra tabla contendrán valores NULL. Cuando haya una coincidencia entre las tablas, la fila completa del conjunto de resultados contendrá los valores de datos de las tablas base.

Para retener la información que no coincida al incluir las filas no coincidentes en los resultados de una combinación, utilice una combinación externa completa. *SQL Server* 2008 proporciona el operador de combinación externa completa, *FULL JOIN*, que incluye todas las filas de ambas tablas, con independencia de que la otra tabla tenga o no un valor coincidente. En forma práctica, podemos decir que *FULL JOIN* es una combinación de *LEFT JOIN* y *RIGHT JOIN*.

Ejemplo: Mostrar los pedidos registrados por los productos, incluya los productos que aun no se ha registrado en algún pedido.

```
USE NEGOCIOS2011
GO

SELECT PD.*, PO.NOMPRODUCTO
FROM VENTAS.PEDIDOSDETA PD FULL JOIN COMPRAS.PRODUCTOS PO
ON PD.IDPRODUCTO = PO.IDPRODUCTO
GO
```

D. COMBINACION CRUZADA

Las combinaciones cruzadas presentan el producto cartesiano de todos los registros de las dos tablas. Se emplea el CROSS JOIN cuando se quiere combinar todos los registros de una tabla con cada registro de otra tabla.

Por ejemplo, elaborar una lista de los productos donde asigne a cada producto todos los posibles proveedores registrados en la base de datos. Aplique combinación cruzada:

```
USE NEGOCIOS2011
GO

SELECT P.IDPRODUCTO, P.NOMPRODUCTO, PR.NOMPROVEEDOR
FROM COMPRAS.PRODUCTOS P CROSS JOIN COMPRAS.PROVEEDORES PR
GO
```

1.3.2. Datos Agrupados

Los resultados de consultas se pueden resumir, agrupar y ordenar utilizando funciones agregadas y las cláusulas **GROUP BY**, **HAVING** y **ORDER BY** con la instrucción **SELECT**. También, se puede usar la cláusula compute (una extensión Transact-SQL) con funciones agregadas para generar un informe con filas detalladas y resumidas.

1.3.2.1. Funciones Agregadas

Las funciones agregadas calculan valores sumarios a partir de datos de una columna concreta. Las funciones agregadas se pueden aplicar a todas las filas de una tabla, a un subconjunto de la tabla especificada por una cláusula **WHERE** o a uno o más grupos de filas de la tabla. De cada conjunto de filas al que se aplica una **función agregada** se genera un solo valor.

A continuación detallamos la sintaxis y resultados de las funciones agregadas:

Función Agregada	Resultado
`Sum([all|distinct] expresión)`	Retorna la suma total de los valores (distintos) de la expresión o columna
`Avg ([all|distinct] expresión)`	Retorna el promedio de los valores (distintos) de la expresión o columna
`Count ([all|distinct] expresión)`	Retorna el número de valores (distintos) no nulos de la expresión
`Count(*)`	Numero de filas seleccionadas
`Max(expresión)`	Retorna el máximo valor de la expresión o columna

`Min(expresión)`	Retorna el mínimo valor de la expresión o columna

Las funciones **SUM y AVG** sólo pueden utilizarse con columnas numéricas: *int* , *smallint* , *tinyint, decimal, numeric, float* y *Money.* Las funciones **MIN** y **MAX** no pueden usarse con tipos de datos *bit* .
Las funciones agregadas distintas de **COUNT(*)** no pueden utilizarse con los tipos de datos *text* e *image*

1.3.2.1.1. Uso de la función COUNT(*)

La función COUNT(*) no requiere ninguna expresión como argumento, porque no emplea información sobre alguna columna. Esta función se utiliza para hallar el número total de filas de una tabla.

Ejemplo: Mostrar la cantidad de pedidos registrados en el año 2011

```
USE NEGOCIOS2011
GO

SELECT COUNT(*) AS 'CANTIDAD DE PEDIDOS'
FROM VENTAS.PEDIDOSCABE
WHERE DATEPART(YY,FECHAPEDIDO)=2011
GO
```

La palabra clave **DISTINCT** es opcional con SUM**,** AVG y COUNT**,** y no se permite con MIN**,** MAX ni COUNT **(*).** Si utiliza **DISTINCT**, el argumento no puede incluir una expresión aritmética, sólo debe componerse de un nombre de columna, esta palabra clave aparece entre paréntesis y antes del nombre de la columna.

Ejemplo: Mostrar la cantidad de clientes que han generado pedidos.

```
USE NEGOCIOS2011
GO

```

```
SELECT COUNT(DISTINCT IDCLIENTE) AS 'NUMERO DE CLIENTES'
FROM VENTAS.PEDIDOSCABE
WHERE DATEPART(YY,FECHAPEDIDO)=1996
GO
```

1.3.2.1.2. Uso de la función AVG

La función AVG () calcula la media aritmética de un conjunto de valores en un campo específico de la consulta
La media calcula por la función AVG es la media aritmética (la suma de los valores dividido por el número de valores).
La función AVG no incluye ningún campo NULL en el cálculo.

Ejemplo: Mostrar el precio Promedio de los productos.

```
USE NEGOCIOS2011
GO

SELECT AVG(PRECIOUNIDAD) AS 'PRECIO PROMEDIO'
FROM COMPRAS.PRODUCTOS
GO
```

1.3.2.1.3. Uso de la función MAX() y MIN()

La función MAX (expr) y la función MIN (expr) devuelven el máximo o mínimo valor de un conjunto de valores contenidos en un campo específico de una consulta.
La expresión (expr) es el campo sobre el que se desea realizar el cálculo; expr pueden incluir el nombre de un campo de una tabla, una constante o una función (la cual puede ser intrínseca o definida por el usuario pero no otras de las funciones agregadas de SQL).

Ejemplo: Mostrar el máximo y el mínimo precio de los productos.

```
USE NEGOCIOS2011
GO

SELECT MAX(PRECIOUNIDAD) AS 'MAYOR PRECIO', MIN(PRECIOUNIDAD)
AS 'MENOR PRECIO'
FROM COMPRAS.PRODUCTOS
GO
```

1.3.2.1.4. Uso de la función SUM

La función SUM (expr) retorna la suma del conjunto de valores contenido en un campo específico de una consulta.
La expresión (*expr*) representa el nombre del campo que contiene los datos que desean sumarse o una expresión que realiza un cálculo utilizando los datos de dichos campos.

Ejemplo: Mostrar la suma de los pedidos registrados en este año.

```
USE NEGOCIOS2011
GO

SELECT SUM(PRECIOUNIDAD*CANTIDAD) AS 'SUMA'
FROM VENTAS.PEDIDOSDETA PD JOIN VENTAS.PEDIDOSCABE PC
ON PD.IDPEDIDO = PC.IDPEDIDO
WHERE YEAR(FECHAPEDIDO)=2011
GO
```

1.3.2.2. Cláusula GROUP BY

Agrupa un conjunto de filas seleccionado en un conjunto de filas de resumen por los valores de una o más columnas o expresiones de SQL Server 2008.

La cláusula **GROUP BY** se utiliza en las instrucciones **SELECT** para dividir la salida de una tabla en grupos. Puede formar grupos según uno o varios nombres de columna, o

según los resultados de las columnas calculadas utilizando tipos de datos numéricos en una expresión. El número máximo de columnas o expresiones es 16.

La cláusula **GROUP BY** aparece casi siempre en instrucciones que también incluyen funciones agregadas, en cuyo caso el agregado genera un valor para cada grupo. A estos valores se les llama **agregados vectoriales.** Un **agregado escalar** es un solo valor generado por una función agregada sin una cláusula **GROUP BY.**

Los valores sumarios (agregados vectoriales) generados por las instrucciones **SELECT** con agregados y una cláusula **GROUP BY** aparecen como columnas en cada fila de los resultados.

Ejemplo: Mostrar la suma y la cantidad de pedidos registrados por cada cliente.

```
USE NEGOCIOS2011
GO

SELECT C.NOMCLIENTE AS 'CLIENTE',
        COUNT(*) AS 'CANTIDAD',  SUM(PRECIOUNIDAD*CANTIDAD) AS
     'SUMA'
FROM VENTAS.PEDIDOSDETA PD JOIN VENTAS.PEDIDOSCABE PC
ON PD.IDPEDIDO = PC.IDPEDIDO
JOIN VENTAS.CLIENTES C ON C.IDCLIENTE = PC.IDCLIENTE
GROUP BY C.NOMCLIENTE
GO
```

Si incluye la clausula WHERE en una consulta agregada, ésta se aplica antes de calcular el valor o la función agregada.

Ejemplo: Mostrar la suma de pedidos registrados por cada cliente en el año 1996.

```
USE NEGOCIOS2011
GO

SELECT     C.NOMCLIENTE AS 'CLIENTE',
           SUM(PRECIOUNIDAD*CANTIDAD) AS 'SUMA'
```

```
FROM VENTAS.PEDIDOSDETA PD JOIN VENTAS.PEDIDOSCABE PC
ON PD.IDPEDIDO = PC.IDPEDIDO
JOIN VENTAS.CLIENTES C ON C.IDCLIENTE = PC.IDCLIENTE
WHERE YEAR(FECHAPEDIDO)=1996
GROUP BY C.NOMCLIENTE
GO
```

1.3.2.3. Cláusula HAVING

Es posible que necesitemos calcular un agregado, pero que no necesitemos obtener todos los datos, solo los que cumplan una condición del agregado. Por ejemplo, podemos calcular el valor de las ventas por producto, pero que solo queramos ver los datos de los productos que hayan vendido más o menos de una determinada cantidad. En estos casos, debemos utilizar la clausula HAVING.

Una vez que GROUP BY ha combinado los registros, HAVING muestra cualquier registro agrupado por la cláusula GROUP BY que satisfaga las condiciones de la cláusula HAVING.

Ejemplo: Mostrar los clientes cuyo importe total de pedidos (suma de pedidos registrados por cada cliente) sea mayor a 1000.

```
USE NEGOCIOS2011
GO

SELECT     C.NOMCLIENTE AS 'CLIENTE',
           SUM(PRECIOUNIDAD*CANTIDAD) AS 'SUMA'
FROM VENTAS.PEDIDOSDETA PD JOIN VENTAS.PEDIDOSCABE PC
ON PD.IDPEDIDO = PC.IDPEDIDO
JOIN VENTAS.CLIENTES C ON C.IDCLIENTE = PC.IDCLIENTE
GROUP BY C.NOMCLIENTE
HAVING SUM(PRECIOUNIDAD*CANTIDAD)>1000
GO
```

Se utiliza la cláusula WHERE para excluir aquellas filas que no desea agrupar y la cláusula HAVING para filtrar los registros una vez agrupados.

Ejemplo: Mostrar los clientes cuyo importe total de pedidos (suma de pedidos registrados por cliente) sea mayor a 1000 siendo registrados en el año 2011.

```
USE NEGOCIOS2011
GO

SELECT     C.NOMCLIENTE AS 'CLIENTE',
           SUM(PRECIOUNIDAD*CANTIDAD) AS 'SUMA'
FROM VENTAS.PEDIDOSDETA PD JOIN VENTAS.PEDIDOSCABE PC
ON PD.IDPEDIDO = PC.IDPEDIDO
JOIN VENTAS.CLIENTES C ON C.IDCLIENTE = PC.IDCLIENTE
WHERE YEAR(FECHAPEDIDO)=2011
GROUP BY C.NOMCLIENTE
HAVING SUM(PRECIOUNIDAD*CANTIDAD)>1000
GO
```

1.3.3. AGREGAR CONJUNTO DE RESULTADOS: UNION

La operación UNION combina los resultados de dos o más consultas en un solo conjunto de resultados que incluye todas las filas que pertenecen a las consultas de la unión. La operación UNION es distinta de la utilización de combinaciones de columnas de dos tablas.

Para utilizar la operación UNION, debemos aplicar algunas reglas básicas para combinar los conjuntos de resultados de dos consultas con UNION:

- El número y el orden de las columnas deben ser idénticos en todas las consultas.
- Los tipos de datos deben ser compatibles.

Sintaxis:

```
  { <consulta> | ( <expresión> ) }
  UNION [ALL]
  <consulta | (<expresión>)
  [UNION [ALL]
<consulta > | ( <expresión> )
    [...n]]
```

Argumentos:

<consulta > | (<expresión>): Es una especificación o expresión de consulta que devuelve datos que se van a combinar con los datos de otra especificación o expresión de consulta. No es preciso que las definiciones de las columnas que forman parte de una operación UNION sean iguales, pero deben ser compatibles a través de una conversión implícita. Cuando los tipos son los mismos, pero varían en cuanto a precisión, escala o longitud, el resultado se determina según las mismas reglas para combinar expresiones.

UNION: Especifica que se deben combinar varios conjuntos de resultados para ser devueltos como un solo conjunto de resultados.

ALL: Agrega todas las filas a los resultados. Incluye las filas duplicadas. Si no se especifica, las filas duplicadas se quitan.

Ejemplos: En el siguiente ejemplo, mostrar los productos que tengan el mayor y menor precio, visualice en ambos casos el nombre del producto.

```
USE NEGOCIOS2011
GO

SELECT NOMPRODUCTO, PRECIOUNIDAD
FROM COMPRAS.PRODUCTOS
WHERE PRECIOUNIDAD = (SELECT MAX(P.PRECIOUNIDAD)
                      FROM COMPRAS.PRODUCTOS P)
UNION
SELECT NOMPRODUCTO, PRECIOUNIDAD
FROM COMPRAS.PRODUCTOS
WHERE PRECIOUNIDAD = (SELECT MIN(P.PRECIOUNIDAD)
                      FROM COMPRAS.PRODUCTOS P)
GO
```

Ejemplos: En el siguiente ejemplo, mostrar la cantidad de pedidos registrados por empleado en el año 2011 y los empleados que no registraron pedidos en el 2011.

```
USE NEGOCIOS2011
GO

SELECT     E.NOMEMPLEADO,
           E.APEEMPLEADO,
           COUNT(*) AS 'CANTIDAD'
FROM RRHH.EMPLEADOS E JOIN VENTAS.PEDIDOSCABE P
ON E.IDEMPLEADO = P.IDEMPLEADO
WHERE YEAR(FECHAPEDIDO)=2011
GROUP BY E.NOMEMPLEADO, E.APEEMPLEADO
UNION
SELECT E.NOMEMPLEADO, E.APEEMPLEADO, 0 AS 'CANTIDAD'
FROM RRHH.EMPLEADOS E
WHERE E.IDEMPLEADO NOT IN(SELECT P.IDEMPLEADO FROM
                          VENTAS.PEDIDOSCABE P
                          WHERE YEAR(FECHAPEDIDO)=2011)
GO
```

1.3.4. RESUMEN DE DATOS: CUBE

El operador CUBE genera un conjunto de resultados que es un cubo multidimensional. Este operador genera filas de agregado mediante la cláusula GROUP BY simple, filas de super agregado mediante la instrucción ROLLUP y filas de tabulación cruzada.

CUBE genera una agrupación para todas las permutaciones de expresiones de la <lista de elementos compuestos>. El número de agrupaciones generado es igual a (2^n), donde n es el número de expresiones de la <lista de elementos compuestos>.

Ejemplo, la siguiente instrucción:

```
USE NEGOCIOS2011
GO

SELECT     C.NOMBRECLIENTE AS 'CLIENTE',
           YEAR(FECHAPEDIDO) AS AÑO,
           SUM(PRECIOUNIDAD*CANTIDAD) AS 'SUMA'
```

```
FROM VENTAS.PEDIDOSDETA PD JOIN VENTAS.PEDIDOSCABE PC
ON PD.IDPEDIDO = PC.IDPEDIDO
JOIN VENTAS.CLIENTES C ON C.IDCLIENTE = PC.IDCLIENTE
GROUP BY CUBE( C.NOMBRECLIENTE, YEAR(FECHAPEDIDO) )
GO
```

Se genera una fila para cada combinación única de valores de (cliente, año), (cliente) y (año), con un subtotal para cada fila y una fila de total general.

1.3.4.1. Agregar permutaciones múltiples: operador CUBE

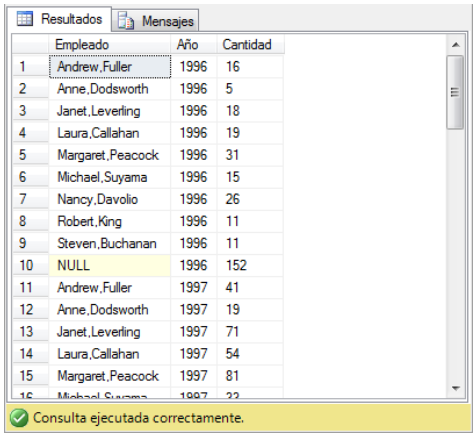
El operador CUBE es una opción adicional de la cláusula GROUP BY en una sentencia SELECT. El operador CUBE se puede aplicar a todas las funciones agregadas, incluidas AVG, SUM, MAX, MIN y COUNT. Se utiliza para producir juegos de resultados que, normalmente, se utilizan para informes de tabulación cruzada. Mientras ROLLUP produce sólo una fracción de las posibles combinaciones subtotales, CUBE produce subtotales para todas las posibles combinaciones de agrupamientos especificados en la cláusula GROUP BY y una suma total.

```
SELECT [column,] group_function(column). . .
FROM table
[WHERE condition]
[GROUP BY  group_by_expression]
[WITH CUBE]
[HAVING having_expression]
[ORDER BY column];
```

El operador CUBE se utiliza con una función agregada para generar filas adicionales en un juego de resultados. Las columnas incluidas en la cláusula GROUP BY tienen referencias cruzadas para producir un super juego de grupos.

Ejemplo: Listar la cantidad de pedidos registrados por cada empleado y año, totalizando la cantidad de pedidos por cada año.

```
USE NEGOCIOS2011
GO

SELECT      E.NOMEMPLEADO+','+E.APEEMPLEADO AS 'EMPLEADO',
            YEAR(P.FECHAPEDIDO) AS 'AÑO',
            COUNT(*) AS 'CANTIDAD'
FROM VENTAS.PEDIDOSCABE P JOIN VENTAS.EMPLEADOS E
ON P.IDEMPLEADO = E.IDEMPLEADO
GROUP BY CUBE( E. NOMEMPLEADO +','+E. APEEMPLEADO,
YEAR(P.FECHAPEDIDO))
GO
```



Resultados | Mensajes

	Empleado	Año	Cantidad
1	Andrew,Fuller	1996	16
2	Anne,Dodsworth	1996	5
3	Janet,Leverling	1996	18
4	Laura,Callahan	1996	19
5	Margaret,Peacock	1996	31
6	Michael,Suyama	1996	15
7	Nancy,Davolio	1996	26
8	Robert,King	1996	11
9	Steven,Buchanan	1996	11
10	NULL	1996	152
11	Andrew,Fuller	1997	41
12	Anne,Dodsworth	1997	19
13	Janet,Leverling	1997	71
14	Laura,Callahan	1997	54
15	Margaret,Peacock	1997	81

Consulta ejecutada correctamente.

Al ejecutar el proceso se puede observar que se lista por cada año y por cada empleado la cantidad de pedidos, visualizando en la línea 10. La suma de la cantidad de pedidos por dicho año, luego se inicializa para el siguiente año.

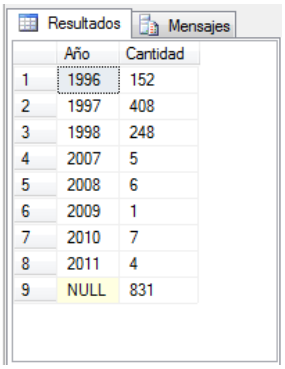
1.3.4.2. RESUMEN DE DATOS: ROLLUP

El operador ROLLUP es útil para generar reportes que contienen subtotales y totales. Genera un conjunto de resultados que es similar al conjunto de resultados del CUBE. Las diferencias entre los operadores CUBE y ROLLUP son las siguientes:

- CUBE genera un conjunto de resultados mostrando agregaciones para todas las combinaciones de valores en las columnas seleccionadas.
- ROLLUP genera un conjunto de resultados mostrando agregaciones para jerarquías en las columnas seleccionadas.

Ejemplo: Mostrar la cantidad de pedidos registrados por cada año. Al finalizar visualice la cantidad total de pedidos.

```
USE NEGOCIOS2011
GO

SELECT      YEAR(FECHAPEDIDO) AS 'AÑO',
            COUNT(*) AS 'CANTIDAD'
FROM VENTAS.PEDIDOSCABE
GROUP BY YEAR(FECHAPEDIDO)
WITH ROLLUP
GO
```

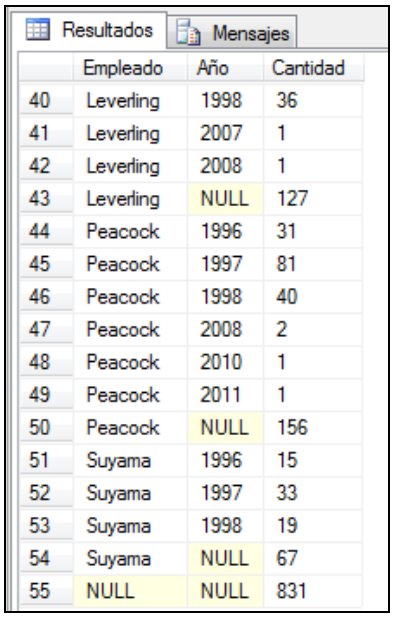

Resultados Mensajes

	Año	Cantidad
1	1996	152
2	1997	408
3	1998	248
4	2007	5
5	2008	6
6	2009	1
7	2010	7
8	2011	4
9	NULL	831

Al usar el operador ROLLUP, se pueden crear agrupamientos en el conjunto de resultados. Para las filas agrupadas, se usa un valor NULL para representar todos los valores para la columna (excepto la columna Sum).
Si se usa un comando SELECT sin el operador ROLLUP, el comando generará los siguientes datos cuando se listen las columnas Ed_nombre, Au_nombre, y Titulo (en ese orden) en la cláusula GROUP_BY.

Ejemplo: Mostrar la cantidad de pedidos registrados por cada empleado y por año, al finalizar visualice la cantidad total de pedidos por cada empleado y la cantidad total de todos los pedidos.

```
USE NEGOCIOS2011
GO

SELECT      E.APELLIDOS AS 'EMPLEADO',
            YEAR(P.FECHAPEDIDO) AS 'AÑO',
            COUNT(*) AS 'CANTIDAD'
FROM TB_PEDIDOSCABE P JOIN TB_EMPLEADOS E
ON P.IDEMPLEADO = E.IDEMPLEADO
GROUP BY E.APELLIDOS, YEAR(P.FECHAPEDIDO)
WITH ROLLUP
GO
```



Resultados | Mensajes

	Empleado	Año	Cantidad
40	Leverling	1998	36
41	Leverling	2007	1
42	Leverling	2008	1
43	Leverling	NULL	127
44	Peacock	1996	31
45	Peacock	1997	81
46	Peacock	1998	40
47	Peacock	2008	2
48	Peacock	2010	1
49	Peacock	2011	1
50	Peacock	NULL	156
51	Suyama	1996	15
52	Suyama	1997	33
53	Suyama	1998	19
54	Suyama	NULL	67
55	NULL	NULL	831

En este caso, visualizamos la cantidad de pedidos por empleado en cada año, totalizando la cantidad de pedidos por empleados (líneas 43, 50 y 54) y el total de todos los pedidos (línea 55).

Resumen

- Un operador es un símbolo que especifica una acción que se realiza en una o más expresiones. Los operadores se clasifican en aritméticos, de asignación, de comparación, lógicos
- Las funciones son métodos que permiten retornar un valor. Las funciones en SQL SERVER se clasifican en: funciones de fechas, de cadena, de conversión.
- La sentencia INSERT agrega una o varias filas a una tabla o vista en SQL SERVER. Utilizando el comando SELECT, podemos agregar múltiples registros.
- La sentencia UPDATE se utiliza para cambiar el contenido de los registros de una o varias columnas de una tabla de la base de datos.
- La sentencia *DELETE* se utiliza para eliminar uno o varios registros de una misma tabla. En una instrucción *DELETE ,* con múltiples tablas, debe incluir el nombre de tabla (Tabla.*). Si especifica más de una tabla para eliminar registros, todas deben tener una relación de muchos a uno.
- La sentencia SELECT recupera las filas de la base de datos y habilita la selección de una o varias filas o columnas de una o varias tablas en SQL Server 2008.
- La instrucción MERGE nos permite realizar múltiples acciones sobre una tabla tomando uno o varios criterios de comparación, es decir, realiza operaciones de inserción, actualización o eliminación en una tabla de destino según los resultados de una combinación con una tabla de origen.
- La sentencia JOIN , en el lenguaje de consulta, permite combinar registros de dos o más tablas en una base de datos relacional. La sentencia JOIN se pueden especificar en las cláusulas *FROM* o *WHERE*, aunque se recomienda que se especifiquen en la cláusula *FROM*. La cláusula **INNER JOIN** permite la combinación de los registros de las tablas, comparando los valores de la columna específica en ambas tablas.
- La sentencia LEFT JOIN retorna la pareja de todos los valores de la izquierda con los valores de la tabla de la derecha correspondientes, o retorna un valor nulo NULL en caso de no correspondencia. El operador *RIGHT JOIN* indica que todas las filas de la segunda tabla se deben incluir en los resultados, con independencia si hay datos coincidentes en la primera tabla.
- Los resultados de consultas se pueden resumir, agrupar y ordenar utilizando funciones agregadas y las cláusulas **GROUP BY**, **HAVING** y **ORDER BY** con la instrucción **SELECT**. También, se puede usar la cláusula compute (una extensión Transact-SQL) con funciones agregadas para generar un informe con filas detalladas y resumidas.

- La operación UNION combina los resultados de dos o más consultas en un solo conjunto de resultados que incluye todas las filas que pertenecen a las consultas de la unión. La operación UNION es distinta de la utilización de combinaciones de columnas de dos tablas
- El operador CUBE genera un conjunto de resultados que es un cubo multidimensional. Este operador genera filas de agregado mediante la cláusula GROUP BY simple, filas de supe agregado mediante la instrucción ROLLUP y filas de tabulación cruzada.
- El operador ROLLUP es útil para generar reportes que contienen subtotales y totales, genera un conjunto de resultados que es similar al conjunto de resultados del CUBE. Las diferencias entre los operadores CUBE y ROLLUP son las siguientes:
  - CUBE genera un conjunto de resultados mostrando agregaciones para todas las combinaciones de valores en las columnas seleccionadas.
  - ROLLUP genera un conjunto de resultados mostrando agregaciones para jerarquías en las columnas seleccionadas.

- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.

  - http://www.sqlmax.com/centro/moduloIII_3.asp?MX=

    Aquí hallará los conceptos de técnicas avanzadas de consulta de datos.
  - http://msdn.microsoft.com/es-es/library/ms180026.aspx

    En esta página, hallará los conceptos de la sentencia UNION.
  - **¡Error! Referencia de hipervínculo no válida.**www.devjoker.com

    Aquí hallará los conceptos de programación TRANSACT SQL.
  - http://manuals.sybase.com/onlinebooks/group-asarc/svs11001/tsqlsp/@Generic__BookTocView/2367;hf=0;pt=2367;lang=es

    En esta página, hallará los conceptos de lenguaje de consulta.

UNIDAD DE APRENDIZAJE
2

PROGRAMACIÓN TRANSACT-SQL

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno escribe rutinas complejas utilizando sentencias TRANSACT-SQL para recupera, inserta, actualiza y elimina información de una base de datos.

TEMARIO

1. Fundamentos de Programación TRANSACT-SQL
2. Construcciones de Programación TRANSACT-SQL
2.1. Variables
2.1.1. Variables Locales
2.1.2. Variables Públicas
2.2. Herramientas para el control de flujos
2.2.1. Estructuras de control IF
2.2.2. Estructura condicional CASE
2.2.2.1. Usar una instrucción SELECT con una expresión CASE de búsqueda
2.2.3. Estructura de control WHILE
2.3. Control de errores en TRANSACT-SQL
2.3.1. Funciones especiales de ERROR
2.3.2. Variable de sistema @@ERROR
2.3.3. Generar un error RAISERROR
2.4. Cursores en TRANSACT-SQL
2.4.1. DECLARE Cursor
2.4.2. Abrir un Cursor: OPEN
2.4.3. Leer un Registro: FETCH
2.4.4. Cerrar el Cursor: CLOSE
2.4.5. Liberar los recursos: DEALLOCATE

ACTIVIDADES PROPUESTAS

- Los alumnos implementan sentencias SQL utilizando estructuras de programación para recuperar y actualizar datos
- Los alumnos implementan sentencias SQL definiendo sentencias de control de errores.
- Los alumnos implementan listados y consultas de datos con totales y subtotales utilizando cursores.

2.1 FUNDAMENTOS DE PROGRAMACION TRANSACT-SQL

SQL es un lenguaje de consulta para los sistemas de bases de datos relaciónales, pero que no posee la potencia de los lenguajes de programación. No permite el uso de variables, estructuras de control de flujo, bucles y demás elementos característicos de la programación. No es de extrañar, **SQL es un lenguaje de consulta, no un lenguaje de programación.**

Sin embargo, SQL es la herramienta ideal para trabajar con bases de datos. Cuando se desea realizar una aplicación completa para el manejo de una base de datos relacional, resulta necesario utilizar alguna herramienta que soporte la capacidad de consulta del SQL y la versatilidad de los lenguajes de programación tradicionales. TRANSACT SQL es el lenguaje de programación que proporciona Microsoft SQL Server para extender el SQL estándar con otro tipo de instrucciones y elementos propios de los lenguajes de programación.

2.1.1 CONSTRUCCION DE PROGRAMACION TRANSACT-SQL

TRANSACT-SQL amplía el SQL estándar con la implementación de estructuras de programación. Estas implementaciones le resultarán familiares a los desarrolladores con experiencia en C++, Java, Visual Basic .NET, C# y lenguajes similares. A continuación, vamos a definir las estructuras de programación implementadas en SQL Server.

2.1.2 VARIABLES

Una variable es una entidad a la que se asigna un valor. Este valor puede cambiar durante el proceso donde se utiliza la variable. *SQL Server* tiene dos tipos de variables: locales y globales. Las variables locales están definidas por el usuario, mientras que las variables globales las suministra el sistema y están predefinidas.

2.1.2.1 Variables Locales

Las variables locales se declaran, nombran y escriben mediante la palabra clave ***declare***, y reciben un valor inicial mediante una instrucción ***select*** **o** ***set***.

Los nombres de las variables locales deben empezar con el símbolo **"@".** A cada variable local se le debe asignar un tipo de dato definido por el usuario o un tipo de dato suministrado por el sistema distinto de *text, image* o *sysname*. Sintaxis:

```
--DECLARA UNA VARIABLE
DECLARE @VARIABLE <TIPO DE DATO>

-- ASIGNA VALOR A UNA VARIABLE
SET @VARIABLE= VALOR
```

En el ejemplo siguiente, declaramos una variable y le asignamos un valor, la cual será utilizada en una clausula WHERE.

```
DECLARE @PRECIO DECIMAL
SET @PRECIO = 50
SELECT * FROM COMPRA.PRODUCTOS
WHERE P.PRECIOUNIDAD >  @PRECIO
GO
```

Podemos utilizar la instrucción SELECT en lugar de la instrucción SET. Una instrucción SELECT utilizada para asignar valores a una o más variables se denomina SELECT de asignación. Si utilizamos el SELECT de asignación, no puede devolver valores al cliente como un conjunto de resultados.

En el ejemplo siguiente, declaramos dos variables y le asignamos el máximo y mínimo precio desde la tabla Compras.productos.

```
USE NEGOCIOS2011
GO

DECLARE @MX DECIMAL, @MN DECIMAL
SELECT     @MX=MAX(PRECIOUNIDAD),
           @MN=MIN(PRECIOUNIDAD)
FROM COMPRAS.PRODUCTOS
```

```
-- IMPRIMIR LOS VALORES DE LAS VARIABLES
PRINT 'MAYOR PRECIO:'+STR(@MX)
PRINT 'MENOR PRECIO:'+STR(@MN)
GO
```

2.1.2.2 Variables Públicas

Las variables globales son variables predefinidas suministradas por el sistema. Se distinguen de las variables locales por tener dos símbolos **"@".** Estas son algunas variables globales del servidor:

Variable	Contenido
@@ERROR	Contiene 0 si la última transacción se ejecutó de forma correcta; en caso contrario, contiene el último número de error generado por el sistema. La variable global @@error se utiliza generalmente para verificar el estado de error de un proceso ejecutado.
@@IDENTITY	Contiene el último valor insertado en una columna ***IDENTITY*** mediante una instrucción *insert*
@@VERSION	Devuelve la Versión del *SQL Server*
@@SERVERNAME	Devuelve el Nombre del Servidor
@@LANGUAGE	Devuelve el nombre del idioma en uso
@@MAX_CONNECTIONS	Retorna la cantidad máxima de conexiones permitidas

En este ejemplo, mostramos la información de algunas variables públicas:

```
--LA VERSION DEL SQL SERVER
PRINT 'VERSION:' + @@VERSION

--LENGUAJE DEL APLICATIVO
PRINT 'LENGUAJE:' + @@LANGUAGE

--NOMBRE DEL SERVIDOR
PRINT 'SERVIDOR:' + @@SERVERNAME
```

```
--NUMERO DE CONEXIONES PERMITIDAS
PRINT 'CONEXIONES:' + STR(@@MAX_CONNECTIONS)
```

2.2 HERRAMIENTAS PARA EL CONTROL DE FLUJOS

El lenguaje de control de flujo se puede utilizar con instrucciones interactivas, en lotes y en procedimientos almacenados. El control de flujo y las palabras clave relacionadas y sus funciones son las siguientes:

Palabra Clave	Función
IF … ELSE	Define una ejecución condicional, cuando la condición la condición es verdadera y la alternativa (else) cuando la condición es falsa
CASE	Es la forma más sencilla de realizar operaciones de tipo IF-ELSE IF-ELSE IF-ELSE. La estructura CASE permite evaluar una expresión y devolver un valor alternativo
WHILE	Estructura repetitiva que ejecuta un bloque de instrucciones mientras la condición es verdadera
BEGIN … END	Define un bloque de instrucciones. El uso del BEGIN…END permite ejecutar un bloque o conjunto de instrucciones.
DECLARE	Declara variables locales
BREAK	Sale del final del siguiente bucle *while* más interno
…CONTINUE	Reinicia del bucle *while*
RETURN [n]	Sale de forma incondicional, suele utilizarse en procedimientos almacenados o desencadenantes. Opcionalmente, se puede definir un numero entero como estado devuelto, que puede asignarse al ejecutar el procedimiento almacenado
PRINT	Imprime un mensaje definido por el usuario o una variable local en la pantalla del usuario
*/*COMENTARIO*/*	Inserta un comentario en cualquier punto de una instrucción *SQL*
--COMENTARIO	Inserta una línea de comentario en cualquier punto de una instrucción *SQL*

2.2.1 Estructuras de control IF

La palabra clave ***IF*** se utiliza para definir una condición que determina si se ejecutará la instrucción siguiente. La instrucción *SQL* se ejecuta si la condición se cumple, es decir, si devuelve *TRUE* (verdadero). La palabra clave **ELSE** introduce una instrucción SQL alternativa que se ejecuta cuando la condición **IF** devuelva *FALSE*. La sintaxis de la estructura condicional IF:

```
IF (<expression>)
  BEGIN
        ...
  END
ELSE IF (<expression>)
  BEGIN
        ...
  END
ELSE
  BEGIN
        ...
  END
```

Ejemplo: Visualice un mensaje donde indique si un empleado (ingrese su codigo) ha realizado pedidos.

```
DECLARE @IDEMP INT, @CANTIDAD INT
SET @IDEMP = 6
--RECUPERAR LA CANTIDAD DE PEDIDOS DEL EMPLEADO DE CODIGO 6
SELECT @CANTIDAD = COUNT(*)
FROM VENTAS.PEDIDOSCABE WHERE IDEMPLEADO = @IDEMP
--EVALUA EL VALOR DE CANTIDAD
IF  @CANTIDAD = 0
      PRINT 'EL EMPLEADO NO HA REALIZADO ALGUN PEDIDO'
ELSE IF @CANTIDAD = 1
      PRINT 'HA REGISTRADO 1 PEDIDO, CONTINUE TRABAJANDO'
ELSE
       PRINT 'HA REGISTRADO PEDIDOS'
GO
```

Ejemplo: utilizamos la estructura IF para evaluar la existencia de un registro; si existe actualizamos los datos de la tabla; si no existe (ELSE) insertamos el registro.

```
DECLARE @COPAIS VARCHAR(3), @NOMBRE VARCHAR(50)
SET @COPAIS = '99'
SET @NOMBRE = 'ESPAÑA'
--EVALUA SI EXISTE EL REGISTRO DE LA TABLA, SI EXISTE
ACTUALIZO, SINO INSERTO
IF EXISTS(SELECT * FROM TB_PAISES WHERE IDPAIS = @COPAIS)
  BEGIN
      UPDATE TB_PAISES
      SET NOMBREPAIS = @NOMBRE
      WHERE IDPAIS = @COPAIS
  END
ELSE
  BEGIN
      INSERT INTO TB_PAISES VALUES (@COPAIS, @NOMBRE)
  END
GO
```

2.2.2 Estructura condicional CASE

La estructura CASE evalúa una lista de condiciones y devuelve una de las varias expresiones de resultado posibles. La expresión CASE tiene dos formatos:

- La expresión CASE sencilla compara una expresión con un conjunto de expresiones sencillas para determinar el resultado.
- La expresión CASE buscada evalúa un conjunto de expresiones booleanas para determinar el resultado.

Ambos formatos admiten un argumento ELSE opcional. La sintaxis del CASE:

```
    CASE <expresión>
        WHEN <valor_expresion> THEN <valor_devuelto>
        WHEN <valor_expresion1> THEN <valor_devuelto1>
        ELSE <valor_devuelto2>  -- Valor por defecto
    END
```

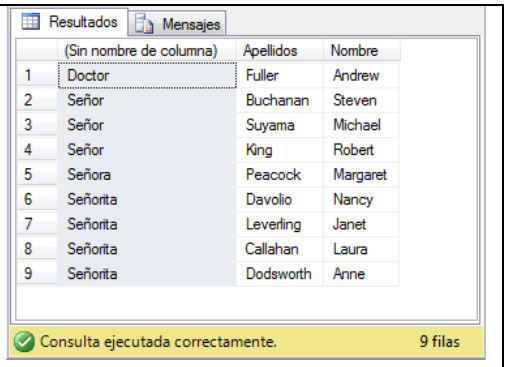
Ejemplo: Declare una variable donde le asigne el numero del mes, evalúe el valor de la variable y retorne el mes en letras.

```
DECLARE @M INT, @MES VARCHAR(20)
SET @M=4
SET @MES = (CASE @M
                WHEN 1      THEN 'ENERO'
                WHEN  2     THEN  'FEBRERO'
                WHEN  3     THEN  'MARZO'
                WHEN  4     THEN  'ABRIL'
                WHEN  5     THEN  'MAYO'
                WHEN  6     THEN  'JUNIO'
                WHEN  7     THEN  'JULIO'
                WHEN  8     THEN  'AGOSTO'
                WHEN  9     THEN  'SETIEMBRE'
                WHEN  10    THEN  'OCTUBRE'
                WHEN  11    THEN  'NOVIEMBRE'
                WHEN  12    THEN  'DICIEMBRE'
                ELSE 'NO ES MES VALIDO'
            END)
PRINT @MES
```

La estructura CASE se puede utilizar en cualquier instrucción o cláusula que permite una expresión válida. Por ejemplo, puede utilizar CASE en instrucciones como SELECT, UPDATE, DELETE y SET, y en cláusulas como select_list, IN, WHERE, ORDER BY y HAVING. La función CASE es una expresión especial de Transact SQL que permite que se muestre un valor alternativo dependiendo de una columna. Este cambio es temporal, con lo que no hay cambios permanentes en los datos.

Ejemplo: Mostrar los datos de los empleados evaluando el valor del campo tratamiento asignando, para cada valor, una expresión.

```
USE NEGOCIOS2011;
GO

SELECT (CASE TRATAMIENTO
        WHEN 'SRTA.' THEN 'SEÑORITA'
        WHEN 'SR.' THEN 'SEÑOR'
        WHEN 'DR.' THEN 'DOCTOR'
        WHEN 'SRA.' THEN 'SEÑORA'
        ELSE 'NO TRATAMIENTO'
      END),APELLIDOS, NOMBRE
FROM RRHH.EMPLEADOS
ORDER BY 1;
GO
```



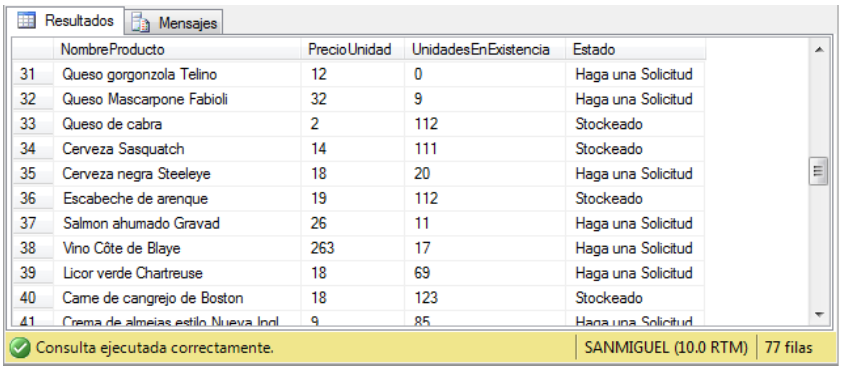
2.2.2.1 Usar una instrucción SELECT con una expresión CASE de búsqueda

En una instrucción SELECT, la expresión CASE de búsqueda permite sustituir valores en el conjunto de resultados basándose en los valores de comparación.

En el ejemplo siguiente, listamos los datos de los productos y definimos una columna llamada ESTADO, el cual evaluará stock de cada producto imprimiendo un valor: Stockeado, Limite, Haga una solicitud.

```
DECLARE @STOCK INT
SET @STOCK=100

SELECT NOMBREPRODUCTO, PRECIOUNIDAD,
          UNIDADESENEXISTENCIA,
          'ESTADO'= (CASE
            WHEN UNIDADESENEXISTENCIA>@STOCK THEN 'STOCKEADO'
            WHEN UNIDADESENEXISTENCIA=@STOCK THEN 'LIMITE'
            WHEN UNIDADESENEXISTENCIA<@STOCK THEN 'HAGA UNA
SOLICITUD'
          END)
FROM TB_PRODUCTOS
GO
```

Al ejecutar la operación listamos los productos y su estado, tal como se muestra.



Resultados | Mensajes

	NombreProducto	PrecioUnidad	UnidadesEnExistencia	Estado
31	Queso gorgonzola Telino	12	0	Haga una Solicitud
32	Queso Mascarpone Fabioli	32	9	Haga una Solicitud
33	Queso de cabra	2	112	Stockeado
34	Cerveza Sasquatch	14	111	Stockeado
35	Cerveza negra Steeleye	18	20	Haga una Solicitud
36	Escabeche de arenque	19	112	Stockeado
37	Salmon ahumado Gravad	26	11	Haga una Solicitud
38	Vino Côte de Blaye	263	17	Haga una Solicitud
39	Licor verde Chartreuse	18	69	Haga una Solicitud
40	Carne de cangrejo de Boston	18	123	Stockeado
41	Crema de almejas estilo Nueva Inol	9	85	Haga una Solicitud

Consulta ejecutada correctamente. | SANMIGUEL (10.0 RTM) | 77 filas

2.2.3 Estructura de control WHILE

La estructura WHILE ejecuta en forma repetitiva un conjunto o bloque de instrucciones SQL siempre que la condición especificada sea verdadera. Se puede controlar la ejecución de instrucciones en el bucle WHILE con las palabras clave BREAK y CONTINUE.

La sintaxis de *WHILE* es:

```
WHILE <expresion>
    BEGIN
       ...
    END
```

Por ejemplo: Implemente un programa que permita listar los 100 primeros números enteros, visualizando en cada caso si es par o impar.

```
DECLARE @contador int
 SET @contador = 0
 WHILE (@contador < 100)
 BEGIN
      SET @contador = @contador + 1
      IF  @contador %2 =0
            PRINT cast(@contador AS varchar) +' es un Número Par'
      ELSE
            PRINT cast(@contador AS varchar) + ' es un Número Impar'
END
```

Por ejemplo: Listar los 5 primeros registros de la tabla productos.

```
DECLARE @COUNTER INT = 1;
WHILE @COUNTER < 6
BEGIN
  SELECT TOP(1) P.IDPRODUCTO, P.NOMBREPRODUCTO, P.PRECIOUNIDAD
  FROM COMPRAS.PRODUCTOS P
  WHERE IDPRODUCTO = @COUNTER
  SET @COUNTER += 1;
END;
```

BREAK **y** ***CONTINUE*** controlan el funcionamiento de las instrucciones dentro de un bucle *WHILE*. ***BREAK*** permite salir del bucle *WHILE*. ***CONTINUE*** hace que el bucle *WHILE* se inicie de nuevo. La sintaxis de *BREAK* y *CONTINUE* es:

```
WHILE BOOLEAN_EXPRESION
  BEGIN
           EXPRESION_SQL

          [BREAK]
              [EXPRESION_SQL]

          [CONTINUE]
              [EXPRESION_SQL]
 END
```

Por ejemplo: Actualizar las unidades de existencia de los productos asignándoles el valor de 1000 de aquellos productos cuyo stock sea cero.

```
USE NEGOCIOS2011
GO

DECLARE @ID INT, @NOMBRE VARCHAR(50)
WHILE  EXISTS (SELECT *
               FROM TB_PRODUCTOS
               WHERE UNIDADESENEXISTENCIA=0)
```

```
BEGIN
  SELECT TOP 1 @ID= P.IDPRODUCTO, @NOMBRE=P.NOMBREPRODUCTO
  FROM TB_PRODUCTOS P  WHERE UNIDADESENEXISTENCIA=0

  UPDATE TB_PRODUCTOS
  SET UNIDADESENEXISTENCIA=1000
  WHERE IDPRODUCTO=@ID

  PRINT 'PRODUCTO:'+@NOMBRE + ' SE ACTUALIZO EL STOCK'
  CONTINUE
END;
```

2.3 CONTROL DE ERRORES EN TRANSACT-SQL

SQL Server proporciona el control de errores a través de las instrucciones TRY y CATCH. Estas nuevas instrucciones suponen un gran paso adelante en el control de errores en SQL Server. La sintaxis de TRY CATCH es la siguiente:

```
BEGIN TRY
      EXPRESION_SQL
END TRY
BEGIN CATCH
      EXPRESION_SQL
END CATCH
```

Ejemplo: Implemente un programa que evalúa la división de dos números enteros.Si la división ha sido exitosa, imprima un mensaje: NO HAY ERROR. Caso contrario imprimir un mensaje: SE HA PRODUCIDO UN ERROR

```
BEGIN TRY
      DECLARE @DIVISOR INT, @DIVIDENDO INT, @RESULTADO INT
      SET @DIVIDENDO = 100
      SET @DIVISOR = 9
```

```
      -- ESTA LINEA PROVOCA UN ERROR DE DIVISION POR 0
      SET @RESULTADO = @DIVIDENDO/@DIVISOR
      PRINT 'NO HAY ERROR'
END TRY
BEGIN CATCH
      PRINT 'SE HA PRODUCIDO UN ERROR'
END CATCH;
```

2.3.1 Funciones especiales de Error

Las funciones especiales de error, están disponibles únicamente en el bloque **CATCH** para la obtención de información detallada del error. A continuación, presentamos las funciones que se utilizan en el control de errores

Error	Descripción
ERROR_NUMBER()	Devuelve el numero de error
ERROR_SEVERITY()	Devuelve la severidad del error
ERROR_STATE()	Devuelve el estado del error
ERROR_PROCEDURE()	Devuelve el nombre del procedimiento almacenado que ha provocado el error
ERROR_LINE()	Devuelve el número de línea en la que se ha producido el error.
ERROR_MESSAGE()	Devuelve el mensaje de error

Ejemplo: Implemente un programa que evalúa la división de dos números enteros. Si la división ha sido exitosa, imprima un mensaje: NO HAY ERROR. Caso contrario imprimir el mensaje de error y el estado del error

```
BEGIN TRY
      DECLARE @DIVISOR INT, @DIVIDENDO INT, @RESULTADO INT
      SET @DIVIDENDO = 100
      SET @DIVISOR = 9
       -- ESTA LINEA PROVOCA UN ERROR DE DIVISION POR 0
       SET @RESULTADO = @DIVIDENDO/@DIVISOR
       PRINT 'NO HAY ERROR'
 END TRY
```

```
BEGIN CATCH
     PRINT ERROR_MESSAGE()
     PRINT ERROR_STATE()
END CATCH;
```

2.3.2 Variable de sistema @@ERROR

Devuelve el número de error de la última instrucción TRANSACT-SQL ejecutada; si la variable devuelve 0, la TRANSACT-SQL anterior no encontró errores.

Ejemplo: Implemente un bloque de instrucciones que permita eliminar un registro de Cliente por su código. Si hay un error en el proceso, visualice un mensaje de error

```
DELETE FROM VENTAS.CLIENTES WHERE IDCLIENTE = 'ALFKI'
IF @@ERROR<>0
     PRINT 'NO SE PUEDE ELIMINAR'
```

Otra forma de implementar este proceso, es controlándolo a través del bloque TRY CATCH.

```
BEGIN TRY
     DELETE FROM TB_CLIENTES
     WHERE IDCLIENTE = 'ALFKI'
END TRY
BEGIN CATCH
     IF @@ERROR=547
     PRINT 'NO SE PUEDE ELIMINAR ESTE CLIENTE'
END CATCH
```

La variable @@ERROR devuelve un número de error que representa el error de la operación. Si el error se encuentra en la vista de catálogo **sys**.**sysmessages**, entonces @@ERROR, contendrá el valor de la columna **sys**.**sysmessages**.**error** para dicho error. Puede ver el texto asociado con el número de error @@ERROR en **sys**.**sysmessages**.**description**

Ejemplo: Implemente un proceso que elimine los productos cuyo valor de sus unidades en existencia, sea menor a 20, en caso de no ejecutar exitosamente el error, implemente el CATCH para controlar el error.

```
BEGIN TRY
      DELETE FROM TB_PRODUCTOS
      WHERE UNIDADESENEXISTENCIA<20
END TRY
BEGIN CATCH
      DECLARE @MENSAJE VARCHAR(255)
      --RECUPERAR LA DESCRIPCION DEL VALOR DE @@ERROR
      SELECT @MENSAJE= M.DESCRIPTION
      FROM SYS.SYSMESSAGES M WHERE M.ERROR=@@ERROR
      PRINT @MENSAJE
END CATCH
```

2.3.3 Generar un error RAISERROR

En ocasiones, es necesario provocar voluntariamente un error, por ejemplo nos puede interesar que se genere un error cuando los datos incumplen una regla de negocio.
Podemos provocar un error en tiempo de ejecución a través de la función RAISERROR.

La función RAISERROR recibe tres parámetros, el mensaje del error (o código de error predefinido), la severidad y el estado.

La severidad indica el grado de criticidad del error. Admite valores de 0 al 25, pero solo podemos asignar valores del 0 al 18. Los errores el 20 al 25 son considerados fatales por el sistema, y cerraran la conexión que ejecuta el comando RAISERROR. Para asignar valores del 19 al 25, necesitaremos ser miembros de la función de SQL Server **sysadmin**.

En el presente ejercicio, evaluamos los valores de dos variables provocando un error utilizando RAISERROR.

```
DECLARE @TIPO INT, @CLASIFICACION INT
SET @TIPO = 1
SET @CLASIFICACION = 3
IF (@TIPO = 1 AND @CLASIFICACION = 3)
      BEGIN
      RAISERROR ('EL TIPO NO PUEDE VALER UNO Y LA CLASIFICACION 3',
                  16, -- SEVERIDAD
                  1   -- ESTADO
      )
END
```

2.4 CURSORES EN TRANSACT-SQL

Un cursor es una variable que nos permite recorres con un conjunto de resultados obtenidos a través de una sentencia SELECT fila por fila.

El uso de los cursores es una técnica que permite tratar fila por fila el resultado de una consulta, contrariamente al SELECT SQL que trata a un conjunto de fila. Los cursores pueden ser implementador por instrucciones TRANSACT-SQL (cursores ANSI-SQL) o por la API OLE-DB.

Se utilizarán los cursores ANSI cuando sea necesario tratar las filas de manera individual en un conjunto o cuando SQL no pueda actuar únicamente sobre las filas afectadas. Los cursores API serán utilizados por las aplicaciones cliente para tratar volúmenes importantes o para gestionar varios conjuntos de resultados.

Cuando trabajemos con cursores, debemos seguir los siguientes pasos:

- Declarar el cursor, utilizando **DECLARE**
- Abrir el cursor, utilizando **OPEN**
- Leer los datos del cursor, utilizando **FETCH** ... **INTO**
- Cerrar el cursor, utilizando **CLOSE**
- Liberar el cursor, utilizando **DEALLOCATE**

La sintaxis general para trabajar con un cursor es la siguiente:

```
-- DECLARACIÓN DEL CURSOR
DECLARE <NOMBRE_CURSOR> CURSOR
FOR <SENTENCIA_SQL>

-- APERTURA DEL CURSOR
OPEN <NOMBRE_CURSOR>

-- LECTURA DE LA PRIMERA FILA DEL CURSOR
FETCH <NOMBRE_CURSOR> INTO <LISTA_VARIABLES>

WHILE (@@FETCH_STATUS = 0)
BEGIN
       -- LECTURA DE LA SIGUIENTE FILA DE UN CURSOR
       FETCH <NOMBRE_CURSOR> INTO <LISTA_VARIABLES>
       ...
END -- FIN DEL BUCLE WHILE

-- CIERRA EL CURSOR
CLOSE <NOMBRE_CURSOR>

-- LIBERA LOS RECURSOS DEL CURSOR
DEALLOCATE <NOMBRE_CURSOR>
```

2.4.1 Declare CURSOR

Esta instrucción permite la declaración y la descripción del cursor ANSI.

Sintaxis:

```
-- DECLARACIÓN DEL CURSOR
DECLARE <NOMBRE_CURSOR> [INSENSITIVE][SCROLL] CURSOR
FOR <SENTENCIA_SQL>
FOR [READ ONLY | UPDATE[ OF LISTA DE COLUMNAS]]
```

- **INSENSITIVE** solo se permiten las operaciones sobre la fila siguiente
- **SCROLL** los desplazamientos en las filas del cursor podrán hacerse en todos los sentidos.
- **UPDATE** especifica que las actualizaciones se harán sobre la tabla de origen del cursor. Un cursor INSENSITIVE con una cláusula ORDER BY no puede actualizarse. Un cursor con ORDER BY, UNION, DISTINCT o HAVING es INSENSITIVE y READ ONLY.

Además de esta sintaxis conforme con la norma ISO, TRANSACT-SQL propone una sintaxis ampliada en la definición de los cursores:

```
DECLARE <nombre_cursor> [LOCAL | GLOBAL ]
FORWARD_ONLY | SCROLL ]
[STATIC | KEYSET | DYNAMIC | FAST_FOWARD ]
[READ_ONLY | SCROLL_LOCKS | OPTIMISTIC ]
[TYPE_WARNING]
FOR <sentencia_sql>
{FOR UPDATE [ OF lista de columnas ]}
```

- LOCAL el alcance del cursor es local en el lote, el procedimiento o la función en curso de ejecución en el que está definido el curso.
- GLOBAL el alcance del cursor el global a la conexión. La opción de base de datos **default to local cursor** está definido en falso de manera predeterminada.
- FOWARD_ONLY los datos se extraen del cursor por orden de aparición (del primero al último).
- STATIC se realiza una copia temporal de los datos en la base de datos **tempdb** para que el cursor no se vea afectado por las modificaciones que puedan realizarse sobre la base de datos.
- KEYSET las filas y su orden en el cursor se fijan en el momento de la apertura del cursor. Las referencias hacia cada una de estas filas de información se conservan en una tabla temporal en **tempdb**.
- DYNAMIC el cursor refleja exactamente los datos presentes en la base de datos. Esto significa que el número de filas, su orden y su valor pueden variar de forma dinámica.

- FAST_FORWARD permite definir un cursor hacia adelante y como sólo lectura (FORWARD_ONLY y READ_ONLY).
- SCROLL_LOCKS permite garantizar el éxito de las instrucciones UPDATE y DELETE que pueden ser ejecutadas en relación al cursor. Con este tipo de cursor, se fija un bloqueo en la apertura para evitar que otra transacción trate de modificar los datos.
- OPTIMISTIC con esta opción, puede que una operación de UPDATE o DLETE realizada en el cursor no pueda ejecutarse correctamente, porque otra transacción haya modificado los datos en paralelo.
- TYPE_WARNING se envía un mensaje (warning) a la aplicación cliente, si se efectúan conversiones implícitas de tipo.

2.4.2 Abrir un Cursor

Esta instrucción permite hacer operativo el cursor y crear eventualmente las tablas temporales asociadas. La variable @@CURSOR_ROWS se asigna después de OPEN.

Teniendo en cuenta el espacio en disco y la memoria utilizada, y el bloqueo eventual de los datos en la apertura del cursor, esta operación debe ser ejecutada lo más cerca posible del tratamiento de los datos extraídos del cursor.

Sintaxis:

```
OPEN [ GLOBAL ] <nombre_cursor>
```

2.4.2.1 Leer un Registro: FETCH

Es la instrucción que permite extraer una fila del cursor y asignar valores a variables con su contenido. Tras FETCH, la variable @@FETCH_STATUS está a 0 si FETCH no retorna errores.

Sintaxis:

```
FETCH [NEXT | PRIOR | LAST | ABSOLUTE n | RELATIVE n ]
      [FROM] [GLOBAL ] <nombre del cursor>
      [INTO lista de variables ]
NEXT
```

- NEXT lee la fila siguiente (única opción posible para los INSENSITIVE CURSOR).
- PRIOR lee la fila anterior
- FIRST lee la primera fila
- LAST lee la última fila
- ABSOLUTE n lee la enésima fila del conjunto
- RELATIVE n lee la enésima fila a partir de la fila actual.

Cuando trabajamos con cursores, la función @@FETCH_STATUS nos indica el estado de la última instrucción FETCH emitida. Los valores posibles son los siguientes:

Valor devuelto	Descripción
0	La instrucción FETCH se ejecutó correctamente
-1	La instrucción FETCH no se ejecutó correctamente o la fila estaba más allá del conjunto de resultados.
-2	Falta la fila recuperada.

2.4.3 Cerrar el cursor

Cierra el cursor y libera la memoria. Esta operación debe interponerse lo antes posible con el fin de liberar los recursos cuanto antes. La sintaxis es la siguiente:

```
CLOSE <nombre_cursor>
```

Después de cerrado el cursor, ya no es posible capturarlo o actualizarlo/eliminarlo. Al cerrar el cursor se elimina su conjunto de claves dejando la definición del cursor intacta, es decir, un cursor cerrado puede volver a abrirse sin tener que volver a declararlo.

2.4.3.1 Liberar los recursos

Es un comando de limpieza, no forma parte de la especificación ANSI. La sintaxis es la siguiente:

```
DEALLOCATE  <nombre_cursor>
```

Ejemplo 1: Trabajando con un cursor, mostrar el primer registro de productos

```
--CURSORES
DECLARE MI_CURSOR CURSOR FOR
      SELECT TOP 1 * FROM COMPRAS.PRODUCTOS

--ABRIR
OPEN MI_CURSOR

--IMPRIMIR EL PRIMER REGISTRO
FETCH NEXT  FROM MI_CURSOR

--CERRAR
CLOSE MI_CURSOR

--LIBERAR
DEALLOCATE MI_CURSOR
```

Ejemplo 2: Trabajando con un cursor dinámico, defina un cursor dinámico que permita visualizar: el primer registro, el registro en la posición 6 y el último registro.

```
--CURSORES
DECLARE MI_CURSOR CURSOR SCROLL FOR
      SELECT   * FROM COMPRAS.PRODUCTOS

-- ABRIR
OPEN MI_CURSOR
```

```
-- IMPRIMIR LOS REGISTROS
FETCH FIRST FROM MI_CURSOR
FETCH ABSOLUTE 6  FROM MI_CURSOR
FETCH LAST FROM MI_CURSOR

-- CERRAR
CLOSE MI_CURSOR

-- LIBERAR
DEALLOCATE MI_CURSOR
```

Ejemplo: Trabajando con un cursor, listar los clientes registrados en la base de datos, incluya el nombre del país.

```
-- DECLARO VARIABLES DE TRABAJO
DECLARE @ID VARCHAR(5), @NOMBRE VARCHAR(50), @PAIS VARCHAR(50)

-- DECLARO EL CURSOR
DECLARE MI_CURSOR CURSOR  FOR
      SELECT      C.IDCLIENTE,
                  C.NOMBRECLIENTE,
                  P.NOMBREPAIS
      FROM VENTAS.CLIENTES C
      JOIN VENTAS.PAISES P ON C.IDPAIS=P.IDPAIS

-- ABRIR
OPEN MI_CURSOR

-- LEER EL PRIMER REGISTRO
FETCH MI_CURSOR INTO @ID, @NOMBRE, @PAIS

-- MIENTRAS PUEDA LEER EL REGISTRO
WHILE @@FETCH_STATUS=0
      BEGIN
            --IMPRIMIR EL REGISTRO
            PRINT @ID + ','+@NOMBRE+','+@PAIS
            --LEER EL REGISTRO SIGUIENTE
            FETCH MI_CURSOR INTO @ID, @NOMBRE, @PAIS
      END
```

```
-- CERRAR
CLOSE MI_CURSOR

-- LIBERAR
DEALLOCATE MI_CURSOR;
GO
```

Ejemplo: Trabajando con un cursor, listar la relación de los clientes que han registrado pedidos. En dicho proceso, debemos imprimir el nombre del cliente, la cantidad de pedidos registrados y al finalizar totalizar el proceso (suma total de todos los pedidos).

```
-- DECLARO VARIABLES DE TRABAJO
DECLARE  @NOMBRE VARCHAR(50), @Q INT, @TOTAL INT
SET @TOTAL=0

-- DECLARO EL CURSOR
DECLARE MI_CURSOR CURSOR  FOR
      SELECT      C.NOMBRECIA,
                  COUNT(*)
      FROM VENTAS.CLIENTES C
      JOIN VENTAS.PEDIDOSCABE PC ON C.IDCLIENTE=PC.IDCLIENTE
      GROUP BY C.NOMBRECLIENTE

-- ABRIR
OPEN MI_CURSOR

-- LEER EL PRIMER REGISTRO
FETCH MI_CURSOR INTO @NOMBRE, @Q

-- MIENTRAS PUEDA LEER EL REGISTRO
WHILE @@FETCH_STATUS=0
      BEGIN
      -- IMPRIMIR EL REGISTRO
      PRINT @NOMBRE+','+CAST(@Q AS VARCHAR)
      -- ACUMULAR
      SET @TOTAL += @Q
      -- LEER EL REGISTRO SIGUIENTE
      FETCH MI_CURSOR INTO @NOMBRE, @Q
      END
```

```
-- CERRAR
CLOSE MI_CURSOR

-- LIBERAR
DEALLOCATE MI_CURSOR;

-- IMPRIMIR
PRINT 'TOTAL DE PEDIDOS:' + CAST(@TOTAL AS VARCHAR)

GO
```

Ejemplo: Trabajando con un cursor, imprimir un listado de los pedidos realizados por cada año. En dicho proceso, listar por cada año los pedidos registrados y totalizando dichos pedidos por dicho año.

```
-- DECLARO VARIABLES DE TRABAJO
DECLARE  @Y INT, @Y1 INT, @PEDIDO INT, @MONTO DECIMAL, @TOTAL
DECIMAL

SET @TOTAL=0

-- DECLARO EL CURSOR
DECLARE MI_CURSOR CURSOR FOR
SELECT     YEAR(FECHAPEDIDO),
           PC.IDPEDIDO,
           SUM(PRECIOUNIDAD*CANTIDAD)
FROM VENTAS.PEDIDOSCABE C
JOIN VENTAS.PEDIDOSDETA D ON C.IDPEDIDO=D.IDPEDIDO
GROUP BY YEAR(FECHAPEDIDO), PC.IDPEDIDO
ORDER BY 1

-- ABRIR EL CURSOR
OPEN MI_CURSOR

-- LEER EL PRIMER REGISTRO
FETCH MI_CURSOR INTO @Y, @PEDIDO, @MONTO
```

```
-- ASIGNAR A LA VARIABLE @Y1 EL VALOR INICIAL DE @Y
SET @Y1 = @Y

-- IMPRIMIR EL PRIMER AÑO
PRINT 'AÑO:' + CAST(@Y1 AS VARCHAR)

-- MIENTRAS PUEDA LEER EL REGISTRO
WHILE @@FETCH_STATUS=0
      BEGIN
      -- SI COINCIDEN LOS VALORES ACUMULAR EL TOTAL
      IF(@Y = @Y1)
            SET @TOTAL += @MONTO
      ELSE
            -- SI NO COINCIDEN IMPRIMIR EL TOTAL, INICIALIZAR
            VARIABLES
            BEGIN
            PRINT 'IMPORTE EN:' +CAST(@Y1 AS VARCHAR) +
            SPACE(2)+ 'ES ' +CAST(@TOTAL AS VARCHAR)
            PRINT 'AÑO:' + CAST(@Y AS VARCHAR)
            SET @Y1=@Y
            SET @TOTAL=@MONTO
            END
      -- IMPRIMIR EL REGISTRO
      PRINT CAST(@PEDIDO AS VARCHAR) + SPACE(5)+STR(@MONTO)
      -- LEER EL REGISTRO SIGUIENTE
      FETCH MI_CURSOR INTO @Y, @PEDIDO, @MONTO
      END

-- CERRAR
CLOSE MI_CURSOR

-- LIBERAR
DEALLOCATE MI_CURSOR;

-- IMPRIMIR LOS TOTALES FINALES
PRINT 'IMPORTE EN:'+CAST(@Y1 AS VARCHAR)+SPACE(2)+'ES
'+STR(@TOTAL)
```

Para actualizar los datos de un cursor, debemos especificar la cláusula FOR UPDATE después de la sentencia SELECT en la declaración del cursor, y WHERE CURRENT OF <nombre_cursor> en la sentencia UPDATE.

En el ejemplo siguiente, actualizamos el precio de los productos: si su stock es mayor o igual a 1000, se descuenta el precio al 50%, sino se descuenta el precio al 20%.

```
-- DECLARACION DE VARIABLES PARA EL CURSOR
DECLARE @ID INT,  @NOMBRE VARCHAR(255), @PRECIO DECIMAL,    @ST INT

-- DECLARACIÓN DEL CURSOR DE ACTUALIZACION
DECLARE CPRODUCTO CURSOR FOR
      SELECT      IDPRODUCTO,
                  NOMBREPRODUCTO,
                  PRECIOUNIDAD,
                  UNIDADESENEXISTENCIA
      FROM TB_PRODUCTOS
FOR UPDATE

-- APERTURA DEL CURSOR
OPEN CPRODUCTO

-- LECTURA DE LA PRIMERA FILA DEL CURSOR
FETCH CPRODUCTO INTO @ID, @NOMBRE, @PRECIO, @ST

-- MIENTRAS PUEDA LEER EL REGISTRO
WHILE (@@FETCH_STATUS = 0 )
BEGIN
      IF(@ST>=1000)
            SET @PRECIO = 0.5*@PRECIO
      ELSE
            SET @PRECIO = 0.80*@PRECIO

      UPDATE TB_PRODUCTOS     SET PRECIOUNIDAD = @PRECIO
      WHERE CURRENT OF CPRODUCTO

      --IMPRIMIR
      PRINT 'EL PRECIO DE PRODUCTO '+ @NOMBRE+ ' ES ' + STR(@PRECIO)
```

```
      -- LECTURA DE LA SIGUIENTE FILA DEL CURSOR
      FETCH CPRODUCTO INTO @ID, @NOMBRE, @PRECIO, @ST
END

-- CIERRE DEL CURSOR
CLOSE CPRODUCTO

-- LIBERAR LOS RECURSOS
DEALLOCATE CPRODUCTO
```

Resumen

- SQL es un lenguaje de consulta para los sistemas de bases de datos relaciónales, pero que no posee la potencia de los lenguajes de programación. No permite el uso de variables, estructuras de control de flujo, bucles y demás elementos característicos de la programación.
- TRANSACT-SQL amplía el SQL estándar con la implementación de estructuras de programación. Estas implementaciones le resultarán familiares a los desarrolladores con experiencia en C++, Java, Visual Basic .NET, C# y lenguajes similares.
- Una variable es una entidad a la que se asigna un valor. Este valor puede cambiar durante el proceso donde se utiliza la variable. *SQL Server* tiene dos tipos de variables: locales y globales. Las variables locales están definidas por el usuario, mientras que las variables globales las suministra el sistema y están predefinidas.
- La estructura condicional IF se utiliza para definir una condición que determina si se ejecutará la instrucción siguiente. La instrucción *SQL* se ejecuta si la condición se cumple, es decir, si devuelve *TRUE* (verdadero). La palabra clave **ELSE** introduce una instrucción SQL alternativa que se ejecuta cuando la condición **IF** devuelva *FALSE*.
- La estructura CASE evalúa una lista de condiciones y devuelve una de las varias expresiones de resultado posibles. La expresión CASE tiene dos formatos:
  - La expresión CASE sencilla compara una expresión con un conjunto de expresiones sencillas para determinar el resultado.
  - La expresión CASE buscada evalúa un conjunto de expresiones booleanas para determinar el resultado.
- La estructura WHILE ejecuta en forma repetitiva un conjunto o bloque de instrucciones SQL siempre que la condición especificada sea verdadera. Se puede controlar la ejecución de instrucciones en el bucle WHILE con las palabras clave BREAK y CONTINUE.
- ***BREAK* y *CONTINUE*** controlan el funcionamiento de las instrucciones dentro de un bucle *WHILE*. ***BREAK*** permite salir del bucle *WHILE*. ***CONTINUE*** hace que el bucle *WHILE* se inicie de nuevo.

- SQL Server proporciona el control de errores a través de las instrucciones TRY y CATCH. Estas nuevas instrucciones suponen un gran paso adelante en el control de errores en SQL Server. La variable @@ERROR devuelve el número de error de la última instrucción TRANSACT-SQL ejecutada; si la variable devuelve 0, la TRANSACT-SQL anterior no encontró errores. Si el error se encuentra en la vista de catálogo **sys**.**sysmessages**, entonces @@ERROR contendrá el valor de la columna **sys**.**sysmessages**.**error** para dicho error. Puede ver el texto asociado con el número de error @@ERROR en **sys**.**sysmessages**.**description.**
- Un cursor es una variable que nos permite recorres con un conjunto de resultados obtenidos a través de una sentencia SELECT fila por fila.
- Se utilizarán los cursores ANSI cuando sea necesario tratar las filas de manera individual en un conjunto o cuando SQL no pueda actuar únicamente sobre las filas afectadas. Los cursores API serán utilizados por las aplicaciones cliente para tratar volúmenes importantes o para gestionar varios conjuntos de resultados
- Cuando trabajemos con cursores, debemos seguir los siguientes pasos:
  - Declarar el cursor, utilizando **DECLARE**
  - Abrir el cursor, utilizando **OPEN**
  - Leer los datos del cursor, utilizando **FETCH** ... **INTO**
  - Cerrar el cursor, utilizando **CLOSE**
  - Liberar el cursor, utilizando **DEALLOCATE**
- Para actualizar los datos de un cursor debemos especificar la cláusula FOR UPDATE después de la sentencia SELECT en la declaración del cursor, y WHERE CURRENT OF <nombre_cursor> en la sentencia UPDATE.

- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.

  - http://www.devjoker.com/contenidos/catss/240/Cursores-en-Transact-SQL.aspx
    Aquí hallará los conceptos de cursores TRANSACT SQL.
  - http://www.a2sistemas.com/blog/2009/02/06/uso-de-cursores-en-transact-sql/
    En esta página, hallará los conceptos y ejercicios de Cursores
  - http://www.devjoker.com/gru/Tutorial-Transact-SQL/TSQL/Tutorial-Transact-SQL.aspx
    Aquí hallará los conceptos de programación TRANSACT SQL.

UNIDAD DE APRENDIZAJE
3

PROGRAMACIÓN AVANZADA TRANSACT SQL

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno, haciendo uso de las estructuras de programación TRANSACT-SQL desarrolladas en clase, define e implementa objetos de base de datos que permite realizar operaciones de consulta y actualización de datos.

TEMARIO

1. Funciones definidas por el usuario
1.1. Funciones escalares
1.2. Funciones de tabla en línea
1.3. Funciones de tabla de multi sentencias
1.4. Limitaciones
2. Procedimientos almacenados
2.1. Especificar parámetros
2.2. Uso de cursores en procedimientos almacenados
2.3. Modificar datos con procedimientos almacenados
2.4. Transacciones en TRANSACT-SQL
2.4.1. Transacciones implícitas y explícitas
3. Triggers o disparadores
3.1. Definición de trigger
3.2. Creación de disparadores
3.2.1. Funcionamiento de los disparadores
3.3. Uso de INSTEAD OF

ACTIVIDADES PROPUESTAS

- Los alumnos programan objetos de base de datos para recuperar datos.
- Los alumnos programan objetos de base de datos utilizando TRANSACT-SQL para recuperar y actualizar datos.

3.1 PROGRAMACIÓN AVANZADA TRANSACT SQL

3.1.1 Funciones definidas por el usuario

Las funciones son rutinas que permiten encapsular sentencias TRANSACT-SQL que se ejecutan frecuentemente.

Las funciones definidas por el usuario, en tiempo de ejecución de lenguaje TRANSACT-SQL o común (CLR), acepta parámetros, realiza una acción, como un cálculo complejo, y devuelve el resultado de esa acción como un valor. El valor de retorno puede ser un escalar (único) valor o una tabla.

Las funciones de usuario son definidas para crear una rutina reutilizables que se pueden utilizar en las siguientes maneras:

- En TRANSACT-SQL como SELECT
- En las aplicaciones de llamar a la función
- En la definición de otra función definida por el usuario
- Para parametrizar una vista o mejorar la funcionalidad de una vista indizada
- Para definir una columna en una tabla
- Para definir una restricción CHECK en una columna
- Para reemplazar a un procedimiento almacenado

Las funciones de usuario, según el tipo de retorno se clasifican en las siguientes:

1. Funciones Escalares
2. Funciones con valores de tabla de varias instrucciones
3. Funciones con valores de tabla en línea

3.1.1.1 Funciones escalares

Son aquellas funciones donde retornan un valor único: tipo de datos como int, Money, varchar, real, etc. Pueden ser utilizadas en cualquier lugar, incluso, incorporada dentro de las sentencias SQL.

```
CREATE FUNCTION [PROPIETARIO.] NOMBRE_FUNCION
([{@PARAMETER  TIPO DE DATO [=DEFAULT]} [,..N]])
RETURNS VALOR_ESCALAR
[AS]
BEGIN
       CUERPO DE LA FUNCION
       RETURN EXPRESION_ESCALAR
END
```

Ejemplo: Crear una función que retorne el precio promedio de todos los productos

```
CREATE FUNCTION DBO.PRECIOPROMEDIO() RETURNS DECIMAL
AS
BEGIN
      DECLARE @PROM DECIMAL
      SELECT @PROM=AVG(PRECIOUNIDAD)
      FROM COMPRAS.PRODUCTOS
      RETURN @PROM
END
GO

-- MOSTRAR EL RESULTADO
PRINT DBO.PRECIOPROMEDIO()
```

Ejemplo: Defina una función donde ingrese el id del empleado y retorne la cantidad de pedidos registrados en el presente año

```
CREATE FUNCTION DBO.PEDIDOSEMPLEADO(@ID INT) RETURNS DECIMAL
AS
BEGIN
      DECLARE @Q DECIMAL=0
      SELECT @Q=COUNT(*)
      FROM VENTAS.PEDIDOSCABE
      WHERE YEAR(FECHAPEDIDO)=YEAR(GETDATE()) AND IDEMPLEADO=@ID
      IF @Q IS NULL
            SET @Q=0
      RETURN @Q
END
GO
```

```
-- MOSTRAR EL RESULTADO DEL EMPLEADO DE CODIGO 4
PRINT DBO.PEDIDOSEMPLEADO(4)
```

3.1.1.2 Funciones de tabla en línea

Las funciones de tabla en línea son las funciones que devuelven la salida de una simple declaración SELECT. La salida se puede utilizar adentro de JOINS o querys como si fuera una tabla de estándar.

La sintaxis para una función de tabla en línea es como sigue:

```
CREATE FUNCTION [propietario.] nombre_funcion
([{ @parameter  tipo de dato [ = default]} [,..n]])
RETURNS TABLE
[AS]
        RETURN [(] Sentencia SQL [)]
```

Ejemplo: Defina una función que liste los registros de los clientes, e incluya el nombre del Pais.

```
CREATE FUNCTION DBO.CLIENTES()
RETURNS TABLE
AS
RETURN (SELECT    IDCLIENTE AS 'CODIGO',
                  NOMBRECIA AS 'CLIENTE',
                  DIRECCION,
                  NOMBREPAIS AS 'PAIS'
        FROM VENTAS.CLIENTES C JOIN VENTAS.PAISES P
        ON C.IDPAIS = P.IDPAIS)
GO

-- EJECUTANDO LA FUNCION
SELECT * FROM DBO.CLIENTES() WHERE PAIS='CHILE'
GO
```

Ejemplo: Defina una función que liste los registros de los pedidos por un determinado año, incluya el nombre del producto, el precio que fue vendido y la cantidad vendida

```
CREATE FUNCTION DBO.PEDIDOSAÑO(@Y INT)
RETURNS TABLE
AS
RETURN (SELECT  PC.IDPEDIDO AS 'PEDIDO',
                FECHAPEDIDO,
                NOMBREPRODUCTO,
                PD.PRECIOUNIDAD AS '¨PRECIO',
                CANTIDAD
        FROM VENTAS.PEDIDOSCABE PC
        JOIN VENTAS.PEDIDOSDETA PD ON PC.IDPEDIDO=PD.IDPEDIDO
        JOIN COMPRAS.PRODUCTOS P ON PD.IDPRODUCTO=P.IDPRODUCTO
        WHERE YEAR(FECHAPEDIDO) = @Y)
GO

-- EJECUTANDO LA FUNCION
SELECT * FROM DBO.PEDIDOSAÑO(2010)
GO
```

3.1.1.3 Funciones de tabla de multisentencias

Son similares a los procedimientos almacenados excepto que vuelven una tabla. Este tipo de función se usa en situaciones donde se requiere más lógica y proceso. Lo que sigue es la sintaxis para unas funciones de tabla de multisentencias:

```
CREATE FUNCTION [propietario.] nombre_funcion
([{@parameter  tipo de dato [ = default]} [,..n]])
RETURNS TABLE
[AS]
BEGIN
       Cuerpo de la función
       RETURN
END
```

Ejemplo: Defina una función que retorne el inventario de los productos registrados en la base de datos.

```
CREATE FUNCTION DBO.INVENTARIO()
RETURNS @TABLA TABLE(IDPRODUCTO INT,
                     NOMBRE VARCHAR(50),
                     PRECIO DECIMAL,
                     STOCK INT)
AS
BEGIN
      INSERT INTO @TABLA
      SELECT IDPRODUCTO,
             NOMBREPRODUCTO,
             PRECIOUNIDAD,
             UNIDADESENEXISTENCIA
      FROM COMPRAS.PRODUCTOS
      RETURN
END
GO

-- EJECUTANDO LA FUNCION
SELECT * FROM DBO.INVENTARIO()
GO
```

Ejemplo: Defina una función que permita generar un reporte de ventas por empleado, en un determinado año. En este proceso, la función debe retornar: los datos del empleado, la cantidad de pedidos registrados y el monto total por empleado

```
CREATE FUNCTION DBO.REPORTEVENTAS(@Y INT)
RETURNS @TABLA TABLE(ID INT,
                     NOMBRE VARCHAR(50),
                     CANTIDAD INT,
                     MONTO DECIMAL)
AS
BEGIN
      INSERT INTO @TABLA
      SELECT E.IDEMPLEADO,
             APELLIDOS,
             COUNT(*),
```

```
            SUM(PRECIOUNIDAD*CANTIDAD)
      FROM VENTAS.PEDIDOSCABE PC JOIN VENTAS.PEDIDOSDETA PD
      ON PC.IDPEDIDO = PD.IDPEDIDO JOIN VENTAS.EMPLEADOS E
      ON E.IDEMPLEADO = PC.IDEMPLEADO
      WHERE YEAR(FECHAPEDIDO) = @Y
      GROUP BY E.IDEMPLEADO, APELLIDOS
      RETURN
END
GO

-- IMPRIMIR EL REPORTE DEL AÑO 2010
SELECT * FROM DBO.REPORTEVENTAS(2010)
GO
```

3.1.1.4 Limitaciones

Las funciones definidas por el usuario tienen algunas restricciones. No todas las sentencias SQL son válidas dentro de una función. Las listas siguientes enumeran las operaciones válidas e inválidas de las funciones:

Válido:

- Las sentencias de asignación
- Las sentencias de Control de Flujo
- Sentencias SELECT y modificación de variables locales
- Operaciones de cursores sobre variables locales Sentencias INSERT, UPDATE, DELETE con variables locales

Inválidas:

- Armar funciones no determinadas como GetDate()
- Sentencias de modificación o actualización de tablas o vistas
- Operaciones CURSOR FETCH que devuelven datos del cliente

3.1.2 Procedimientos Almacenados

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 el fracaso del mismo 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 lotes de instrucciones *SQL* en que están precompilados. La primera vez que se ejecuta un procedimiento, el procesador de consultas *SQL Server* lo analiza y prepara un plan de ejecución que se almacena en forma definitiva en una tabla de 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.

Los procedimientos almacenados se crean con ***CREATE PROCEDURE.*** Para ejecutar un procedimiento almacenado, ya sea un procedimiento del sistema o uno definido por el usuario, use el comando ***EXECUTE***. También, puede utilizar el nombre del procedimiento almacenado solo, siempre que sea la primera palabra de una instrucción o lote.

Sintaxis para crear un procedimiento almacenado:

```
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
    -- Añadir parámetros al procedimiento almacenado
    <@Param1> <Datatype_For_Param1> = <Default_Value_For_Param1>,
    <@Param2> <Datatype_For_Param2> = <Default_Value_For_Param2>
AS
BEGIN
    -- Insertar la sentencia para el procedimiento
    Sentencia SQL
END
```

Sintaxis para modificar un procedimiento almacenado:

```
ALTER PROCEDURE NOMBRE_PROCEDIMIENTO
    <@Param1> <Datatype_For_Param1> = <Default_Value_For_Param1>,
    <@Param2> <Datatype_For_Param2> = <Default_Value_For_Param2>
AS
CONSULTA_SQL
```

Sintaxis para eliminar un procedimiento almacenado:

```
DROP PROCEDURE NOMBRE_PROCEDIMIENTO
```

Por ejemplo: Defina un procedimiento almacenado que liste todos los clientes

```
-- PROCEDIMIENTO ALMACENADO
CREATE PROCEDURE USP_CLIENTES
AS
SELECT IDCLIENTE AS CODIGO,
       NOMBRECIA AS CLIENTE,
       DIRECCION,
       TELEFONO
FROM VENTAS.CLIENTES
GO
```

Como se aprecia, el procedimiento mostrado no tiene parámetros de entrada y para ejecutarlo deberá usar una de las siguientes sentencias:

```
-- EJECUTANDO EL PROCEDIMIENTO ALMACENADO
EXEC USP_CLIENTES
GO
```

O simplemente

```
-- EJECUTANDO EL PROCEDIMIENTO ALMACENADO
USP_CLIENTES
GO
```

Por ejemplo: Cree un procedimiento almacenado que permita buscar los datos de los pedidos registrados en una determinada fecha. El procedimiento deberá definir un parámetro de entrada de tipo DateTime

```
CREATE PROCEDURE USP_PEDIDOSFECHAS
@F1 DATETIME
AS
      SELECT *
      FROM VENTAS.PEDIDOSCABE
      WHERE FECHAPEDIDO = @F1
GO
```

Como se aprecia, el procedimiento mostrado tiene 1 parámetro de entrada y para ejecutarlo deberá usar la siguiente sentencia:

```
-- EJECUTANDO EL PROCEDIMIENTO ALMACENADO
EXEC USP_PEDIDOSBYFECHAS @F1='10-01-1996'
GO
```

O simplemente, colocar la lista de los valores el cual será asignada a cada parámetro, donde el primer valor le corresponde a @f1

```
-- EJECUTANDO EL PROCEDIMIENTO ALMACENADO
EXEC USP_PEDIDOSBYFECHAS '10-01-1996'
GO
```

La sentencia ***ALTER PROCEDURE*** permite modificar el contenido del procedimiento almacenado. En este procedimiento, realizamos la consulta de pedidos entre un rango de dos fechas.

```
ALTER PROCEDURE USP_PEDIDOSBYFECHAS
@F1 DATETIME,
@F2 DATETIME
AS
      SELECT *
      FROM VENTAS.PEDIDOSCABE
      WHERE FECHAPEDIDO BETWEEN @F1 AND @F2
GO
```

Para ejecutar el procedimiento almacenado

```
EXEC USP_PEDIDOSBYFECHAS @F1='10-01-1996', @F2='10-10-1996'
GO
```

Para eliminar un procedimiento almacenado, ejecute la instrucción DROP PROCEDURE

```
DROP PROCEDURE USP_PEDIDOSBYFECHAS
GO
```

3.1.2.1 Especificar parámetros

Un procedimiento almacenado se comunica con el programa que lo llama mediante sus parámetros. Cuando un programa ejecuta un procedimiento almacenado, es posible pasarle valores mediante los parámetros del procedimiento.

Estos valores se pueden utilizar como variables estándar en el lenguaje de programación TRANSACT-SQL. El procedimiento almacenado también puede devolver valores al programa que lo llama mediante parámetros OUTPUT. Un procedimiento almacenado puede tener hasta 2.100 parámetros, cada uno de ellos con un nombre, un tipo de datos, una dirección y un valor predeterminado

3.1.2.2 Especificar el nombre del parámetro

Cada parámetro de un procedimiento almacenado, debe definirse con un nombre único. Los nombres de los procedimientos almacenados deben empezar por un solo carácter @, como una variable estándar de TRANSACT-SQL, y deben seguir las reglas definidas para los identificadores de objetos. El nombre del parámetro se puede utilizar en el procedimiento almacenado para obtener y cambiar el valor del parámetro.

3.1.2.3 Especificar la dirección del parámetro

La dirección de un parámetro puede ser de entrada, que indica que un valor se pasa al parámetro de entrada de un procedimiento almacenado o de salida, que indica que el procedimiento almacenado devuelve un valor al programa que lo llama mediante un parámetro de salida. El valor predeterminado es un parámetro de entrada.
Para especificar un parámetro de salida, debe indicar la palabra clave OUTPUT en la definición del parámetro del procedimiento almacenado. El programa que realiza la llamada también debe utilizar la palabra clave OUTPUT al ejecutar el procedimiento almacenado, a fin de guardar el valor del parámetro en una variable que se pueda utilizar en el programa que llama.

3.1.2.4 Especificar un valor de parámetro predeterminado

Puede crear un procedimiento almacenado con parámetros opcionales especificando un valor predeterminado para los mismos. Al ejecutar el procedimiento almacenado, se utilizará el valor predeterminado si no se ha especificado ningún otro.
Es necesario especificar valores predeterminados, ya que el sistema devuelve un error si en el procedimiento almacenado no se especifica un valor predeterminado para un parámetro y el programa que realiza la llamada no proporciona ningún otro valor al ejecutar el procedimiento.

Por ejemplo: Cree un procedimiento almacenado que muestre los datos de los pedidos, los productos que fueron registrados por cada pedido, el precio del producto y la cantidad registrada por un determinado cliente y año. El procedimiento recibirá como parámetro de entrada el código del cliente y el año. Considere que el parámetro año tendrá un valor por defecto: el año del sistema.

```
CREATE PROCEDURE USP_PEDIDOSCLIENTEAÑO
@ID VARCHAR(5),
@AÑO INT = 2011
AS
      SELECT PC.IDPEDIDO AS 'PEDIDO',
             FECHAPEDIDO, NOMBREPRODUCTO,
             PD.PRECIOUNIDAD AS '¨PRECIO',
             CANTIDAD
      FROM VENTAS.PEDIDOSCABE PC JOIN VENTAS.PEDIDOSDETA PD
      ON PC.IDPEDIDO = PD.IDPEDIDO JOIN COMPRAS.PRODUCTOS P
      ON PD.IDPRODUCTO = P.IDPRODUCTO
      WHERE YEAR(FECHAPEDIDO) = @AÑO
      AND IDCLIENTE = @ID
GO
```

Como el procedimiento almacenado ha definido un valor por defecto al parámetro @año, podemos ejecutar el procedimiento enviando solamente el valor para el parámetro @id

```
EXEC USP_PEDIDOSCLIENTEAÑO @ID='ALFKI'
GO
```

O simplemente enviando los valores a los dos parámetros

```
EXEC USP_PEDIDOSCLIENTEAÑO @ID='ALFKI', @AÑO=1997
GO
```

Por ejemplo: Implemente un procedimiento almacenado que retorne la cantidad de pedidos y el monto total de pedidos, registrados por un determinado empleado (parámetro de entrada su id del empleado) y en determinado año (parámetro de entrada). Dicho procedimiento retornará la cantidad de pedidos y el monto total

```
CREATE PROCEDURE USP_REPORTEPEDIDOSEMPLEADO
@ID INT,
@Y INT,
@Q INT OUTPUT,
@MONTO DECIMAL OUTPUT
AS
      SELECT @Q= COUNT(*),
            @MONTO = SUM(PRECIOUNIDAD*CANTIDAD)
      FROM VENTAS.PEDIDOSCABE PC JOIN VENTAS.PEDIDOSDETA PD
      ON PC.IDPEDIDO = PD.IDPEDIDO
      WHERE IDEMPLEADO =@ID AND YEAR(FECHAPEDIDO) = @Y
GO
```

Al ejecutar el procedimiento almacenado, primero declaramos las variables de retorno y al ejecutar, las variables de retorno se le indicara con la expresión OUTPUT.

```
DECLARE @Q INT, @M DECIMAL
EXEC  USP_REPORTEPEDIDOSEMPLEADO @ID=2,
      @Y=1997,
      @Q=@Q OUTPUT,
      @MONTO=@M OUTPUT
GO

PRINT 'CANTIDAD DE PEDIDOS COLOCADOS:' + STR(@Q)
PRINT 'MONTO PERCIBIDO:'+STR(@M)
GO
```

3.1.2.5 Uso de cursores en procedimientos almacenados

Los cursores son especialmente útiles en procedimientos almacenados. Permiten llevar a cabo la misma tarea utilizando sólo una consulta que, de otro modo, requeriría varias. Sin embargo, todas las operaciones del cursor deben ejecutarse dentro de un solo procedimiento. Un procedimiento almacenado no puede abrir, recobrar o cerrar un cursor que no esté declarado en el procedimiento. El cursor no está definido fuera del alcance del procedimiento almacenado.

Por ejemplo: Implemente un procedimiento almacenado que imprimir cada uno de los registros de los productos, donde al finalizar, visualice el total del inventario (Suma de cantidad de productos)

```
CREATE PROCEDURE USP_INVENTARIO
AS
-- DECLARACION DE VARIABLES PARA EL CURSOR
DECLARE @ID INT,  @NOMBRE VARCHAR(255),   @PRECIO DECIMAL, @ST INT,
@INV INT
SET @INV=0

-- DECLARACIÓN DEL CURSOR
DECLARE CPRODUCTO CURSOR FOR
      SELECT  IDPRODUCTO,
              NOMBREPRODUCTO,
              PRECIOUNIDAD,
              UNIDADESENEXISTENCIA
      FROM COMPRAS.PRODUCTOS

-- APERTURA DEL CURSOR
OPEN CPRODUCTO

-- LECTURA DE LA PRIMERA FILA DEL CURSOR
FETCH CPRODUCTO INTO @ID, @NOMBRE, @PRECIO, @ST
WHILE (@@FETCH_STATUS = 0 )
BEGIN
      -- IMPRIMIR
      PRINT STR(@ID) + SPACE(5) + @NOMBRE + SPACE(5) +
             STR(@PRECIO) + SPACE(5) + STR(@ST)
      -- ACUMULAR
      SET @INV += @ST
      -- LECTURA DE LA SIGUIENTE FILA DEL CURSOR
      FETCH CPRODUCTO INTO @ID, @NOMBRE, @PRECIO, @ST
END

-- CIERRE DEL CURSOR
CLOSE CPRODUCTO

-- LIBERAR LOS RECURSOS
DEALLOCATE CPRODUCTO
```

```
PRINT 'INVENTARIO DE PRODUCTOS:' + STR(@INV)
GO
```

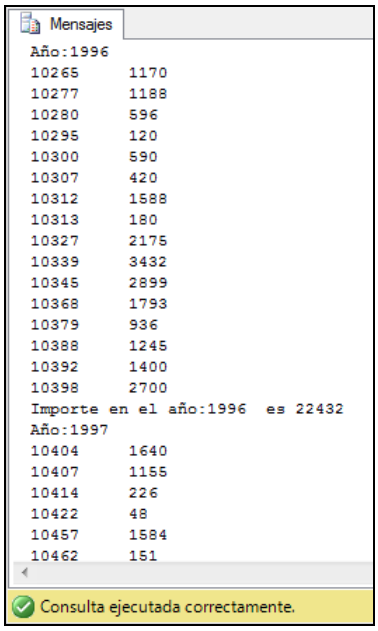
En el siguiente ejemplo, definimos un procedimiemto que permita generar un reporte de los pedidos realizados por un empleado en cada año, totalizando el monto de sus operaciones por cada año.

```
CREATE PROCEDURE USP_REPORTEPEDIDOSXAÑOXEMPLEADO
@EMP INT=1
AS
-- DECLARACIÓN DE VARIABLES DE TRABAJO
DECLARE  @Y INT, @Y1 INT, @PEDIDO INT, @MONTO DECIMAL, @TOTAL DECIMAL
SET @TOTAL=0

-- DECLARACIÓN DEL CURSOR
DECLARE MI_CURSOR CURSOR FOR
      SELECT YEAR(FECHAPEDIDO) AS 'AÑO',
             PC.IDPEDIDO,
             SUM(PRECIOUNIDAD*CANTIDAD) AS MONTO
      FROM VENTAS.PEDIDOSCABE PC
      JOIN VENTAS.PEDIDOSDETA PD
      ON PC.IDPEDIDO=PD.IDPEDIDO
      WHERE IDEMPLEADO = @EMP
      GROUP BY YEAR(FECHAPEDIDO), PC.IDPEDIDO
      ORDER BY 1

-- APERTURA DEL CURSOR
OPEN MI_CURSOR

-- LECTURA DEL PRIMER REGISTRO
FETCH MI_CURSOR INTO @Y, @PEDIDO, @MONTO

-- ASIGNACIÓN DEL VALOR INICIAL DE @Y EN LA VARIABLE @Y1
SET @Y1 = @Y

-- IMPRIMIR EL PRIMER AÑO
PRINT 'AÑO:' + CAST(@Y1 AS VARCHAR)
```

```
-- RECORRER EL CURSOS MIENTRAS HAYAN REGISTROS
WHILE @@FETCH_STATUS=0
      BEGIN
      IF(@Y = @Y1)
      BEGIN
            -- ACUMULAR
            SET @TOTAL += @MONTO
      END
      ELSE
      BEGIN
            PRINT 'AÑO:' + CAST(@Y1 AS VARCHAR) + SPACE(2)+
            'IMPORTE: ' + CAST(@TOTAL AS VARCHAR)
            PRINT 'AÑO:' + CAST(@Y AS VARCHAR)
            SET @Y1=@Y
            SET @TOTAL=@MONTO
            END
            -- IMPRIMIR EL REGISTRO
            PRINT CAST(@PEDIDO AS VARCHAR) + SPACE(5)+
            CAST(@MONTO AS VARCHAR)
      -- LECTURA DEL SIGUIENTE REGISTRO
      FETCH MI_CURSOR INTO @Y, @PEDIDO, @MONTO
      END

-- CERRAR EL CURSOR
CLOSE MI_CURSOR

-- LIBERAR EL RECURSO
DEALLOCATE MI_CURSOR;

PRINT ' AÑO:' + CAST(@Y1 AS VARCHAR) + SPACE(2)+ 'IMPORTE: ' +
STR(@TOTAL)
GO
```

Al ejecutar el procedimiento almacenado, se le envía el parámetro que representa el id del empleado, imprimiendo su record de ventas de pedidos por año

```
USP_REPORTEPEDIDOSXAÑOXEMPLEADO 2
GO
```



Ejecutado el procedimiento almacenado lista cada pedido por año, mostrando al finalizar el total por cada año.

3.1.3 Modificar datos con procedimientos almacenados

Los procedimientos almacenados pueden aceptar datos como parámetros de entrada y pueden devolver datos como parámetros de salida, conjuntos de resultados o valores de retorno. Adicionalmente, los procedimientos almacenados pueden ejecutar sentencias de actualización de datos: INSERT, UPDATE, DELETE

Por ejemplo, defina un procedimiento almacenado para insertar un registro de la tabla Clientes, en este procedimiento, definiremos parámetros de entrada que representan los campos de la tabla.

```
CREATE PROCEDURE USP_INSERTACLIENTE
@ID VARCHAR(5),
@NOMBRE VARCHAR(50),
@DIRECCION VARCHAR(100),
@IDPAIS CHAR(3),
@FONO VARCHAR(15)
AS
INSERT INTO VENTAS.CLIENTES(IDCLIENTE, NOMCLIENTE, DIRECCION,IDPAIS,
TELEFONO)
VALUES(@ID, @NOMBRE, @DIRECCION, @IDPAIS, @FONO)
GO
```

El ejecutar el procedimiento almacenado, se enviará la lista de los parámetros definidos en el procedimiento.

```
EXEC USP_INSERTACLIENTE 'ABCDE', 'JUAN CARLOS MEDINA',
                        'CALLE 25 NO 123','006','5450555'
GO
```

En el siguiente ejemplo, definimos un procedimiento almacenado que permita evaluar la existencia de un registro de empleado para insertar o actualizar sus datos: Si existe el código del empleado, actualice sus datos; sino agregue el registro de empleados

```
CREATE PROCEDURE USP_ACTUALIZAEMPLEADO
@ID INT,
@NOMBRE VARCHAR(50),
@APELLIDO VARCHAR(50),
@FN DATETIME,
@DIRECCION VARCHAR(100),
@IDDIS INT,
@FONO VARCHAR(15),
@IDCARGO INT,
@FC DATETIME
AS
MERGE RRHH.EMPLEADOS AS TARGET
USING
(SELECT @ID, @NOMBRE, @APELLIDO, @FN, @DIRECCION, @IDDIS, @FONO,
@IDCARGO, @FC) AS SOURCE
(IDEMPLEADO, NOMEMPLEADO, APEEMPLEADO, FECNAC, DIRECCION, IDDISTRITO,
FONOEMPLEADO, IDCARGO, FECCONTRATA)
ON (TARGET.IDEMPLEADO = SOURCE.IDEMPLEADO)
WHEN MATCHED THEN
      UPDATE RRHH.EMPLEADOS
      SET   NOMEMPLEADO=@NOMBRE, APEEMPLEADO=@APELLIDO, FECNAC=@FN,
            DIRECCION=@DIRECCION, IDDISTRITO=@IDDIS,
            FONOEMPLEADO=@FONO, IDCARGO=IDCARGO, FECCONTRATA=@FC
WHEN NOT MATCHED THEN
      INSERT  INTO RRHH.EMPLEADOS VALUES(@ID, @NOMBRE, @APELLIDO,
      @FN, @DIRECCION, @IDDIS, @FONO, @IDCARGO, @FC) ;
GO
```

Para este caso, hemos utilizado la instrucción MERGE que evalúa la existencia de un registro. Si existe, actualizará los datos, sino agrega el registro.

En ocasiones, no sabemos si estamos realizando correctamente un proceso de inserción, actualización o eliminación de datos a una tabla(s). El script de T-SQL consiste en realizar un procedimiento almacenado que reciba los datos necesarios para insertarlos en la tabla, para garantizar la ejecución correcta de las actualización utilizo las transacciones "**TRANSACT SQL**" y para validar la reversión de la transacción en caso de que ocurra un **ERROR** utilizo el control de Errores **Try – Catch** con **ROLLBACK**.

3.1.4 TRANSACCIONES EN TRANSACT-SQL

Una transacción es un conjunto de operaciones **TRANSACT SQL** que se ejecutan como un único bloque, es decir, si falla una operación **TRANSACT SQL** fallan todas. Si una transacción tiene éxito, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en una parte permanente de la base de datos. Si una transacción encuentra errores y debe cancelarse o revertirse, se borran todas las modificaciones de los datos.

El ejemplo clásico de transacción es un registro de pedidos, donde agregamos un registro a la tabla pedidoscabe y agregamos un registro a la tabla pedidosdeta (producto solicitado por el cliente) y descontamos el stock del producto solicitado en el pedido.

```
CREATE PROCEDURE USP_AGREGAPEDIDO
-- PARÁMETROS DE PEDIDOSCABE
@IDPED INT,
@IDCLI VARCHAR(5),
@IDEMP INT,
@FECPED DATETIME,

-- PARÁMETROS DE PEDIDOSDETA
@IDPROD INT,
@PRE DECIMAL,
@CANT INT
AS
```

```
      -- AGREGANDO UN REGISTRO A PEDIDOSCABE
      INSERT INTO VENTAS.PEDIDOSCABE(IDPEDIDO,IDCLIENTE,
                  IDEMPLEADO,FECHAPEDIDO)
      VALUES(@IDPED, @IDCLI, @IDEMP, @FECPED)

      -- AGREGANDO UN REGISTRO A PEDIDOSDETA
      INSERT INTO VENTAS.PEDIDOSDETA(IDPEDIDO, IDPRODUCTO,
                  PRECIOUNIDAD,CANTIDAD, DESCUENTO)
      VALUES(@IDPED, @IDPROD, @PRE, @CANT, 0)

      -- DESCONTANDO EL STOCK DE PRODUCTOS
      UPDATE COMPRAS.PRODUCTOS SET UNIDADESENEXISTENCIA -=@CANT
      WHERE IDPRODUCTO = @IDPROD
GO
```

Esta forma de procesar las operaciones de pedidos sería errónea, ya que cada instrucción se ejecutaría y confirmaría de forma independiente, por lo que un error en alguna de las operaciones dejaría los datos erróneos en la base de datos.

3.1.4.1 Transacciones implícitas y explicitas

Para agrupar varias sentencias **TRANSACT SQL** en una única transacción, disponemos de los siguientes métodos:

- ***Transacciones explícitas:*** Cada transacción se inicia explícitamente con la instrucción **BEGIN TRANSACTION** y se termina explícitamente con una instrucción **COMMIT** o **ROLLBACK**.
- ***Transacciones implícitas***: Se inicia automáticamente una nueva transacción cuando se ejecuta una instrucción que realiza modificaciones en los datos, pero cada transacción se completa explícitamente con una instrucción **COMMIT** o **ROLLBACK**.

Sintaxis para el control de errores:

```
BEGIN TRY
    /*Bloque de instrucciones a validar*/
END TRY
```

```
BEGIN CATCH
    /*Bloque de instrucciones que se ejecutan si ocurre  un ERROR*/
END CATCH
```

Sintaxis para el control de las transacciones

```
-- Inicio de transacción con nombre
BEGIN TRAN NombreTransaccion
        /*Bloque de instrucciones a ejecutar en la Transacción*/
COMMIT TRAN NombreTransaccion--Confirmación de la transacción.
ROLLBACK TRAN NombreTransaccion--Reversión de la transacción.
```

En este ejemplo, definimos un procedimiento almacenado que agregue un registro a la tabla de clientes. En este proceso, controlamos la operación a través de una transacción llamada tcliente, evaluamos el proceso con la variable @@ERROR

```
CREATE PROCEDURE  USP_INSERTACLIENTE
@ID VARCHAR(5),
@NOMBRE VARCHAR(50),
@DIRECCION VARCHAR(100),
@IDPAIS CHAR(3),
@FONO VARCHAR(15)
AS
-- INICIO DE LA TRANSACCION
BEGIN TRAN TCLIENTE
      INSERT INTO VENTAS.CLIENTES(IDCLIENTE, NOMCLIENTE, DIRECCION,
                                  IDPAIS, TELEFONO)
      VALUES(@ID, @NOMBRE, @DIRECCION, @IDPAIS, @FONO)
      GO

-- CONTROLAR EL PROCESO
IF @@ERROR = 0
BEGIN
      -- CONFIRMACIÓN DE LA INSERCIÓN
      COMMIT TRAN TCLIENTE
      PRINT 'CLIENTE REGISTRADO'
END
```

```
ELSE
BEGIN
      PRINT @@ERROR
      -- DESHACER LA INSERCIÓN
      ROLLBACK TRAN TCLIENTE
END
GO
```

Para un mejor control de los errores, ejecutamos los procesos dentro del bloque TRY CATCH, donde en caso que las operaciones hayan tenido éxito, ejecutará COMMIT TRAN, pero en caso de error CATCH, ejecutará ROLLBACK TRAN, tal como se muestra en el siguiente ejercicio.

```
CREATE PROCEDURE USP_AGREGAPEDIDO
-- PARÁMETROS DE PEDIDOSCABE
@IDPED INT,
@IDCLI VARCHAR(5),
@IDEMP INT,
@FECPED DATETIME,
-- PARÁMETROS DE PEDIDOSDETA
@IDPROD INT,
@PRE DECIMAL,
@CANT INT
AS
-- INICIO DE LA TRANSACCION
BEGIN TRAN TPEDIDO

-- INICIO DEL CONTROL DE ERRORES
BEGIN TRY

      -- AGREGANDO UN REGISTRO A PEDIDOSCABE
      INSERT INTO
      VENTAS.PEDIDOSCABE(IDPEDIDO,IDCLIENTE,IDEMPLEADO,FECHAPEDIDO)
      VALUES(@IDPED, @IDCLI, @IDEMP, @FECPED)

      -- AGREGANDO UN REGISTRO A PEDIDOSDETA
      INSERT INTO VENTAS.PEDIDOSDETA(IDPEDIDO, IDPRODUCTO,
      PRECIOUNIDAD,CANTIDAD, DESCUENTO)
      VALUES(@IDPED, @IDPROD, @PRE, @CANT, 0)
```

```
      -- DESCONTANDO EL STOCK DE PRODUCTOS
      UPDATE COMPRAS.PRODUCTOS SET UNIDADESENEXISTENCIA -=@CANT
      WHERE IDPRODUCTO = @IDPROD

      -- CONFIRMANDO LA ACTUALIZACION
      COMMIT TRAN TPEDIDO
      PRINT 'PEDIDO REGISTRADO'
END TRY
BEGIN CATCH
      PRINT @@ERROR
      ROLLBACK TRAN TPEDIDO
END CATCH
GO
```

En el siguiente ejercicio, implementamos un procedimiento almacenado para realizar el BACKUP a la tabla ClienteBAK. En este proceso, los registros de la tabla Clientes se irán agregando (si no existen) o actualizando sus datos (si existe el registro) o eliminar el registro en ClienteBAK si éste no existe en Clientes.

```
-- CREAR LA TABLA CLIENTESBAK
CREATE TABLE VENTAS.CLIENTESBAK (
  IDCLIENTE VARCHAR(5) PRIMARY KEY,
  NOMBRECLIENTE VARCHAR(40) NOT NULL,
  DIRECCION VARCHAR(60) NOT NULL,
  IDPAIS CHAR(3),
  TELEFONO VARCHAR(24) NOT NULL
)
GO

-- PROCEDIMIENTO QUE REALIZA BACK UP A LA TABLA CLIENTES
CREATE PROCEDURE USP_CLIENTEBAK
AS
BEGIN TRAN BK
BEGIN TRY
      MERGE VENTAS.CLIENTESBAK AS TARGET
      USING VENTAS.CLIENTES AS SOURCE
      ON (TARGET.IDCLIENTE = SOURCE.IDCLIENTE)
      WHEN MATCHED AND TARGET.NOMBRECLIENTE <> SOURCE.NOMCLIENTE THEN
            UPDATE SET TARGET.NOMBRECLIENTE = SOURCE.NOMCLIENTE,
            TARGET.DIRECCION = SOURCE .DIRCLIENTE, TARGET.IDPAIS =
```

```
            SOURCE.IDPAIS, TARGET.TELEFONO=SOURCE.FONOCLIENTE
      WHEN NOT MATCHED THEN
            INSERT  VALUES(SOURCE.IDCLIENTE,SOURCE.NOMCLIENTE,
            SOURCE.DIRCLIENTE, SOURCE.IDPAIS, SOURCE.FONOCLIENTE)
      WHEN NOT MATCHED BY SOURCE THEN
      DELETE ;
      PRINT 'TRANSACCION COMPLETADA'
      COMMIT TRAN BK
END TRY
BEGIN CATCH
      PRINT @@ERROR
      ROLLBACK TRAN
END CATCH
GO
```

3.1.5 TRIGGERS O DISPARADORES

Los disparadores pueden usarse para imponer la integridad de referencia de los datos en toda la base de datos. Los disparadores también permiten realizar cambios “en cascada” en tablas relacionadas, imponer restricciones de columna más complejas que las permitidas por las reglas, compara los resultados de las modificaciones de datos y llevar a cabo una acción resultante.

3.1.5.1 Definición del disparador

Un disparador es un tipo especial de procedimiento almacenado que **se ejecuta cuando se insertan, eliminan o actualizan datos** de una tabla especificada. Los disparadores pueden ayuda a mantener la integridad de referencia de los datos conservando la consistencia entre los datos relacionados lógicamente de distintas tablas. Integridad de referencia significa que los valores de las llaves primarias y los valores correspondientes de las llaves foráneas deben coincidir de forma exacta.

La principal ventaja de los disparadores es que son automáticos: funcionan cualquiera sea el origen de la modificación de los datos. Cada disparador es específico de una o más operaciones de modificación de datos, ***UPDATE, INSERT o DELETE***. El disparador se ejecuta una vez por cada instrucción.

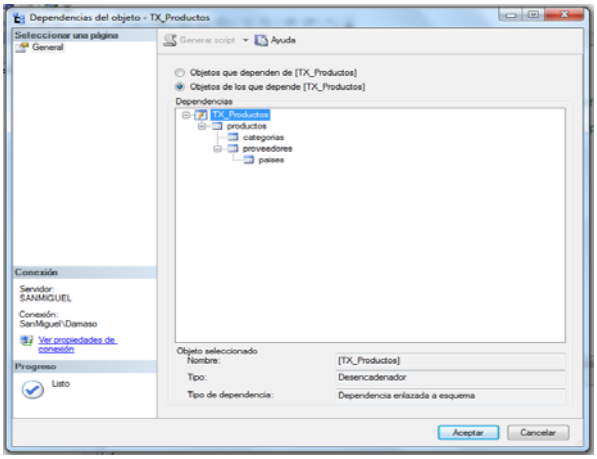
3.1.5.2 Creación de disparadores

Un disparador es un objeto de la base de datos. Cuando se crea un disparador, se especifica la tabla y los comandos de modificación de datos que deben “disparar” o activar el disparador. Luego, se indica la acción o acciones que debe llevar a cabo un disparador.

```
CREATE TRIGGER [ esquema. ]nombre_trigger
ON { Tabla | Vista }
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS
sentencia sql  [ ; ]
```

A continuación, se muestra un ejemplo sencillo. Este disparador imprime un mensaje cada vez que alguien trata de insertar, eliminar o actualizar datos de la tabla Productos

```
CREATE TRIGGER TX_Productos
ON Compras.Productos
FOR INSERT, UPDATE, DELETE
AS
PRINT 'Actualizacion de los registros de Productos'
```



Dependencia del TRIGGER o Disparador en relación a la tabla.

Para modificar el *TRIGGER*, se utiliza la siguiente sintaxis:

```
ALTER TRIGGER TX_PRODUCTOS
ON COMPRAS.PRODUCTOS
FOR INSERT, UPDATE, DELETE
AS
PRINT 'ACTUALIZACION DE LOS REGISTROS DE PRODUCTOS'
```

Para borrar un *TRIGGER*, se utiliza la siguiente sintaxis:

```
DROP TRIGGER TX_PRODUCTOS
```

3.1.5.3 Funcionamiento de los Disparadores

A. Disparador de Inserción

Cuando se inserta una nueva fila en una tabla, *SQL Server* inserta los nuevos valores en la tabla ***INSERTED*** el cual es una tabla del sistema. Está tabla toma la misma estructura del cual se originó el *TRIGGER*, de tal manera que se pueda verificar los datos y ante un error podría revertirse los cambios.

Cree un *TRIGGER* que permita insertar los datos de un Producto siempre y cuando la descripción o nombre del producto sea único.

```
CREATE TRIGGER TX_PRODUCTO_INSERTA
ON COMPRAS.PRODUCTOS
FOR INSERT
AS
IF  (SELECT  COUNT (*) FROM   INSERTED, COMPRAS.PRODUCTOS
      WHERE INSERTED.NOMPRODUCTO = PRODUCTOS.NOMPRODUCTO) >1
BEGIN
      ROLLBACK TRANSACTION
      PRINT 'LA DESCRIPCION DEL PRODUCTO SE ENCUENTRA REGISTRADO'
END
ELSE
      PRINT 'EL PRODUCTO FUE INGRESADO EN LA BASE DE DATOS'
GO
```

En este ejemplo, verificamos el número de productos que tienen la misma descripción y de encontrarse más de un registro de productos no se deberá permitir ingresar los datos del producto. Este disparador imprime un mensaje si la inserción se revierte y otro si se acepta.

B. Disparador de Eliminación

Cuando se elimina una fila de una tabla, *SQL Server* inserta los valores que fueron eliminados en la tabla ***DELETED*** el cual es una tabla del sistema. Está tabla toma la misma estructura del cual se origino el *TRIGGER*, de tal manera que se pueda verificar los datos y ante un error podría revertirse los cambios. En este caso, la reversión de los cambios significará restaurar los datos eliminados.

Cree un *TRIGGER* el cual permita eliminar Clientes los cuales no han registrado algún pedido. De eliminarse algún Cliente que no cumpla con dicha condición la operación no deberá ejecutarse.

```
CREATE TRIGGER TX_ELIMINA_ELIMINA
ON VENTAS.CLIENTES
FOR DELETE
AS
IF EXISTS (SELECT * FROM VENTAS.PEDIDOSCABE
WHERE PEDIDOSCABE.IDCLIENTE = (SELECT IDCLIENTE FROM DELETED) )
BEGIN
      ROLLBACK TRANSACTION
      PRINT 'EL CLIENTE TIENE REGISTRADO POR LO MENOS 1 PEDIDOS'
END
```

En este ejemplo, verificamos si el cliente tiene pedidos registrados, de ser así la operación deberá ser cancelada.

C. Disparador de Actualización

Cuando se actualiza una fila de una tabla, *SQL Server* inserta los valores que antiguos en la tabla ***DELETED*** y los nuevos valores los inserta en la tabla ***INSERTED***. Usando

estas dos tablas se podrá verificar los datos y ante un error podrían revertirse los cambios.

Cree un *TRIGGER* que valide el precio unitario y su Stock de un producto, donde dichos datos sean mayores a cero.

```
CREATE TRIGGER TX_PRODUCTO_ACTUALIZA
ON COMPRAS.PRODUCTOS
FOR UPDATE
AS
IF (SELECT PRECIOUNIDAD  FROM INSERTED) <=0 OR
(SELECT UNIDADESENEXISTENCIA FROM INSERTED)<=0
BEGIN
      PRINT 'EL PRECIO O UNIDADESENEXISTENCIA DEBEN SER MAYOR A CERO'
      ROLLBACK TRANSACTION
END
```

Cree un *TRIGGER* que bloquee actualizar el id del producto en el proceso de actualización.

```
CREATE TRIGGER TX_PRODUCTO_ACTUALIZA_ID
ON COMPRAS.PRODUCTOS
FOR UPDATE
AS
IF UPDATE(IDPRODUCTO)
BEGIN
      PRINT 'NO SE PUEDE ACTUALIZAR EL ID DEL PRODUCTO'
      ROLLBACK TRANSACTION
END
```

3.1.5.4 Uso de INSTEAD OF

Los desencadenadores INSTEAD OF pasan por alto las acciones estándar de la instrucción de desencadenamiento: INSERT, UPDATE o DELETE. Se puede definir un desencadenador INSTEAD OF para realizar comprobación de errores o valores en una o más columnas y, a continuación, realizar acciones adicionales antes de insertar el registro.

Por ejemplo, cuando el valor que se actualiza en un campo de tarifa de una hora de trabajo de una tabla de planilla excede un valor específico, se puede definir un desencadenador para producir un error y revertir la transacción o insertar un nuevo registro en un registro de auditoría antes de insertar el registro en la tabla de planilla.

A. INSTEAD OF INSERT

Se pueden definir desencadenadores INSTEAD OF INSERT en una vista o tabla para reemplazar la acción estándar de la instrucción INSERT. Normalmente, el desencadenador INSTEAD OF INSERT se define en una vista para insertar datos en una o más tablas base.

Las columnas de la lista de selección de la vista pueden o no admitir valores NULL. Si una columna de la vista no admite valores NULL, una instrucción INSERT debe proporcionar los valores para la columna.

```
-- CREAR UNA VISTA QUE LISTE LAS COLUMNAS DE CLIENTES.
CREATE VIEW INSTEADVIEW
AS
SELECT IDCLIENTE,
       NOMCLIENTE,
       DIRCLIENTE,
       IDPAIS,
       FONOCLIENTE
FROM VENTAS.CLIENTES
GO

-- CREAR UN TRIGGER INSTEAD OF INSERT PARA LA VISTA.
CREATE TRIGGER INSTEADTRIGGER ON INSTEADVIEW
INSTEAD OF INSERT
AS
BEGIN
  INSERT INTO VENTAS.CLIENTESBAK
  SELECT IDCLIENTE, NOMCLIENTE, DIRCLIENTE, IDPAIS, FONOCLIENTE
  FROM INSERTED
END
GO
```

B. INSTEAD OF UPDATE

Se pueden definir desencadenadores INSTEAD OF UPDATE en una vista o tabla para reemplazar la acción estándar de la instrucción UPDATE. Normalmente, el desencadenador INSTEAD OF UPDATE se define en una vista para modificar datos en una o más tablas. Las instrucciones UPDATE que hacen referencia a vistas que contienen desencadenadores INSTEAD OF UPDATE deben suministrar valores para todas las columnas que no admiten valores NULL a las que hace referencia la cláusula SET

C. INSTEAD OF DELETE

Se pueden definir desencadenadores INSTEAD OF DELETE en una vista o una tabla para reemplazar la acción estándar de la instrucción DELETE. Normalmente, el desencadenador INSTEAD OF DELETE se define en una vista para modificar datos en una o más tablas. Las instrucciones DELETE no especifican modificaciones de los valores de datos existentes. Las instrucciones DELETE sólo especifican las filas que se van a eliminar. La tabla **inserted** pasada a un desencadenador DELETE siempre está vacía. La tabla **deleted** enviada a un desencadenador DELETE contiene una imagen de las filas en el estado que tenían antes de emitir la instrucción DELETE

3.1.5.5 Restricciones de los disparadores

A continuación, se describen algunas limitaciones o restricciones impuestas a los disparadores por SQL Server:

- Una tabla puede tener un máximo de tres disparadores: uno de actualización, uno de inserción y uno de eliminación.
- Cada disparador puede aplicarse a una sola tabla. Sin embargo, un mismo disparador se puede aplicar a las tres acciones del usuario: *UPDATE, INSERT* y *DELETE*.
- No se puede crear un disparador en una vista ni en una tabla temporal, aunque los disparadores pueden hacer referencia a las vistas o tablas temporales.
- Los disparadores no se permiten en las tablas del sistema. Aunque no aparece ningún mensaje de error su crea un disparador en una tabla del sistema, el disparador no se utilizará.

Resumen

- Las funciones definidas por el usuario, en tiempo de ejecución de lenguaje TRANSACT-SQL o común (CLR), acepta parámetros, realiza una acción, como un cálculo complejo, y devuelve el resultado de esa acción como un valor. El valor de retorno puede ser un escalar (único) valor o una tabla.
- Las funciones escalares son aquellas funciones donde retornan un valor único: tipo de datos como int, Money, varchar, real, etc. Pueden ser utilizadas en cualquier lugar, incluso, incorporada dentro de las sentencias SQL. Las funciones de tabla en línea son las funciones que devuelven la salida de una simple declaración SELECT. La salida se puede utilizar adentro de JOINS o querys como si fuera una tabla de estándar. Las funciones de tabla multisentencias son similares a los procedimientos almacenados excepto que vuelven una tabla
- 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:
    1. Incluir parámetros
    2. Llamar a otros procedimientos
    3. Devolver un valor de estado a un procedimiento de llamada o lote para indicar el éxito o el fracaso del mismo y la razón de dicho fallo.
    4. Devolver valores de parámetros a un procedimiento de llamada o lote
    5. Ejecutarse en *SQL Server* remotos
- Un procedimiento almacenado se comunica con el programa que lo llama mediante sus parámetros. Cuando un programa ejecuta un procedimiento almacenado, es posible pasarle valores mediante los parámetros del procedimiento. Estos valores se pueden utilizar como variables estándar en el lenguaje de programación TRANSACT-SQL. El procedimiento almacenado también puede devolver valores al programa que lo llama mediante parámetros OUTPUT. Un procedimiento almacenado puede tener hasta 2.100 parámetros, cada uno de ellos con un nombre, un tipo de datos, una dirección y un valor predeterminado

- Los procedimientos almacenados pueden aceptar datos como parámetros de entrada y pueden devolver datos como parámetros de salida, conjuntos de resultados o valores de retorno. Adicionalmente, los procedimientos almacenados pueden ejecutar sentencias de actualización de datos: INSERT, UPDATE, DELETE
- Una transacción es un conjunto de operaciones **TRANSACT SQL** que se ejecutan como un único bloque, es decir, si falla una operación **TRANSACT SQL** fallan todas. Si una transacción tiene éxito, todas las modificaciones de los datos realizadas durante la transacción se confirman y se convierten en una parte permanente de la base de datos. Si una transacción encuentra errores y debe cancelarse o revertirse, se borran todas las modificaciones de los datos.
- Los disparadores pueden usarse para imponer la integridad de referencia de los datos en toda la base de datos. Los disparadores también permiten realizar cambios “en cascada” en tablas relacionadas, imponer restricciones de columna más complejas que las permitidas por las reglas, compara los resultados de las modificaciones de datos y llevar a cabo una acción resultante.
- Cuando se inserta una nueva fila en una tabla, *SQL Server* inserta los nuevos valores en la tabla ***INSERTED*** el cual es una tabla del sistema. Está tabla toma la misma estructura del cual se originó el *TRIGGER*, de tal manera que se pueda verificar los datos y ante un error podría revertirse los cambios
- Cuando se elimina una fila de una tabla, *SQL Server* inserta los valores que fueron eliminados en la tabla ***DELETED*** el cual es una tabla del sistema. Está tabla toma la misma estructura del cual se origino el *TRIGGER*, de tal manera que se pueda verificar los datos y ante un error podría revertirse los cambios. En este caso la reversión de los cambios significará restaurar los datos eliminados.
- Cuando se actualiza una fila de una tabla, *SQL Server* inserta los valores que antiguos en la tabla ***DELETED*** y los nuevos valores los inserta en la tabla ***INSERTED***. Usando estas dos tablas se podrá verificar los datos y ante un error podrían revertirse los cambios

- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

  http://www.devjoker.com/contenidos/catss/292/Transacciones-en-Transact-SQL.aspx

  Aquí hallará los conceptos de Transacciones en TRANSACT SQL.

  http://www.sqlmax.com/func2.asp

  En esta página, hallará los conceptos funciones

http://msdn.microsoft.com/es-es/library/ms189260.aspx

Aquí hallará los conceptos de manejo de parámetros en procedimientos almacenados

UNIDAD DE APRENDIZAJE
4

MANEJO DE DATOS XML EN SQL SERVER

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno, haciendo uso de los comandos de consulta y actualización de datos, realizamos operación de recuperación de datos en formato XML.

TEMARIO

1.1. Introducción
1.1.1. Porque utilizar bases de datos relacionales para datos XML
1.1.2. Tipos de datos XML
1.1.3. Integración de relaciones y XML
1.1.4. FOR XML y mejoras OPENXML
1.2. Procesamiento XML en SQL SERVER
1.2.1. Tipos de datos XML
1.2.2. Almacenamiento de datos XML
1.2.3. Recuperación de datos de tipo XML
1.2.3.1. Usar modo RAW
1.2.3.2. Usar modo AUTO
1.2.3.3. Usar modo EXPLICIT
1.2.3.4. Usar modo PATH
1.2.4. Recuperación de datos con OPENXML

ACTIVIDADES PROPUESTAS

- Los alumnos actualizan datos con formato XML.
- Los alumnos recuperan los datos de una base de datos relacional en formato XML.

4.1 INTRODUCCION

Extensible Markup Language (XML) ha sido ampliamente adoptado como un formato independiente de la plataforma de representación de datos. Es útil para el intercambio de información entre los débilmente acoplados, sistemas dispares, como en el negocio a negocio (B2B) y flujos de trabajo. El intercambio de datos ha sido un gran impulsor de las tecnologías XML.

XML es cada vez más presente en las aplicaciones empresariales que se utilizan para el modelado de datos semi-estructurados y no estructurados. Una de estas aplicaciones es la gestión de documentos. Documentos (mensajes de correo electrónico, por ejemplo) son semi-estructuradas por la naturaleza.

Si los documentos se almacenan en un servidor de base de datos como XML, potentes aplicaciones pueden ser desarrolladas, tales como:

- Las aplicaciones que recuperar los documentos en base a su contenido.
- Las aplicaciones que consulta por el contenido parcial, como la búsqueda de la sección cuyo título contiene la palabra "fondo".
- Las aplicaciones que los documentos agregados.

Estos escenarios se están convirtiendo en posibles con el aumento en el desarrollo y la disponibilidad de las aplicaciones que generan y consumen XML. Por ejemplo, el Microsoft Office 2008 System permite a los usuarios generar Microsoft Word, Excel, Visio, InfoPath y documentos de formato XML.

4.1.1 ¿Por qué utilizar bases de datos relacionales para los datos de XML?

El almacenamiento de datos XML en una base de datos relacional proporciona beneficios en las áreas de gestión de datos y el procesamiento de consultas.

Microsoft SQL Server proporciona una consulta de gran alcance y las capacidades de modificación de datos en datos relacionales. En SQL Server 2008, estas capacidades se extienden a consultar y modificar datos XML. Esto permite a su empresa para aprovechar las inversiones hechas en versiones anteriores, como las inversiones en

las áreas de costos basado en optimización y almacenamiento de datos. Por ejemplo, las técnicas de indexación en bases de datos relacionales son bien conocidas. Estos se han extendido a la indexación de XML para que las consultas se puedan optimizar el uso de costos basado en las decisiones.

Los datos XML pueden interactuar con datos relacionales y aplicaciones SQL. Esto significa que XML puede ser introducido en el sistema según las necesidades de modelado de datos se presentan sin interrumpir las aplicaciones existentes. El servidor de base de datos también proporciona una funcionalidad administrativa para la gestión de datos XML (por ejemplo, BACKUP, recuperación y replicación).

La compatibilidad nativa con XML en SQL Server 2008 es necesario para hacer frente a aumentar el uso de XML. Beneficios de la empresa de desarrollo de aplicaciones de la compatibilidad con XML en SQL Server 2008.

En las secciones siguientes, se ofrece una visión general de la compatibilidad con XML en SQL Server 2005 y 2008, describe algunos de los escenarios de motivación para el uso de XML, y analizar en detalle el lado del servidor y de cliente XML conjuntos de características.

4.1.2 Tipos de datos XML

El modelo de datos XML tiene características que lo hacen muy difícil por no decir prácticamente imposible asignar al modelo de datos relacional. De datos XML tiene una estructura jerárquica que puede ser recursivo, bases de datos relacionales ofrecen poco apoyo para los datos jerárquicos (modelados como relaciones de clave externa).

El orden del documento es una propiedad inherente de instancias XML y debe ser preservada en los resultados de la consulta. Esto está en contraste con los datos relacionales, que no está ordenado, el orden debe ser ejecutada con columnas adicionales pedidos. Volver a montar el resultado en la consulta es costoso para los realistas esquemas XML que se descomponen los datos XML en un gran número de tablas.

SQL Server 2008 introduce un tipo de datos nativo llamado XML. Un usuario puede crear una tabla que tiene una o más columnas de tipo XML, además de columnas

relacionales. Las variables y los parámetros de XML también son permitidos. Los valores XML se almacenan en un formato interno como objetos binarios grandes (BLOB) para apoyar a las características del modelo XML, tales como el orden del documento y las estructuras recursivas, con mayor fidelidad.

SQL Server 2008 ofrece colecciones de esquemas XML como una manera de manejar esquemas XML del W3C como metadatos. Un tipo de datos XML se puede asociar con una colección de esquemas XML para hacer cumplir las restricciones de esquema en las instancias XML. Cuando los datos XML se asocian con una colección de esquemas XML, se llama *XML con tipo,* de lo contrario se llama *XML sin tipo.* Con tipo y XML sin tipo se alojan en un marco único, el modelo de datos XML se conserva, y el procesamiento de consultas aplica la semántica XML. La infraestructura relacional subyacente se utiliza ampliamente para este propósito. Es compatible con la interoperabilidad entre datos relacionales y XML, con objeto de alcanzar la adopción más generalizada de las características XML.

4.1.2.1 Integración de Relacionales y XML

Los usuarios pueden almacenar datos relacionales y XML dentro de la misma base de datos. En pocas palabras, el motor de base de datos sabe hacer honor a la modelo de datos XML, además del modelo de datos relacional. De datos relacionales y aplicaciones de SQL, siguen comportándose correctamente en la actualización a SQL Server 2008. De datos XML que residen en los archivos y las columnas de texto o la imagen se puede mover en las columnas de tipo de datos XML en el servidor. La columna XML se puede indexar, consultar y modificar con los métodos de tipo de datos XML.

La base de datos relacional mejora la infraestructura existente y los componentes del motor como el motor de almacenamiento y el procesador de consultas para procesamiento de XML. Por ejemplo, crear índices **XML B$^+$** árboles y los planes de consulta se pueden ver en la salida del plan de presentación. Puesto que la funcionalidad de gestión de datos, tales como copia de seguridad / restauración y replicación, se integra en el marco relacional, esta funcionalidad está disponible en los datos XML. Además, las nuevas características de datos de gestión, como reflejo de base y el aislamiento de instantáneas, el trabajo con el tipo de datos XML para proporcionar una experiencia de usuario sin fisuras.

Los datos estructurados se almacenan en tablas y columnas relacionales. El tipo de datos XML es una elección adecuada de los datos semi-estructurados y marcado con XML cuando la aplicación necesita para llevar a cabo de grano fino de la consulta y modificación de los datos.

4.1.3 FOR XML y mejoras OpenXML

La funcionalidad XML se ha mejorado de varias maneras. Se trabaja sobre casos tipo de datos XML y otros nuevos tipos de SQL, tales como **[n] varchar (max).**

La nueva directiva TYPE genera una instancia de tipo de datos XML que se pueden asignar a una columna de XML, una variable o parámetro, o consultar utilizando los métodos de tipo de datos XML. Esto permite el anidamiento de SELECT... FOR instrucciones de tipo XML.

El modo PATH permite a los usuarios especificar la ruta en el árbol XML donde el valor de una columna debe aparecer y junto con el mencionado anidación. Es más cómodo para escribir que para EXPLÍCITA XML.

El XSINIL Directiva, que se utiliza en conjunción con elementos, mapas NULL a un elemento con el atributo **xsi: nil = "true".** Además, la Directiva permite a la nueva raíz un nodo raíz que se especificarán en todos los modos de FOR XML. La directiva XMLSCHEMA nueva genera un esquema XSD en línea. FOR XML en SQL Server 2008 también permite a los usuarios especificar los nombres de elementos para sustituir a la **<row>** por defecto en modo FOR XML RAW.

Mejoras funcionales OpenXML consisten en la posibilidad de aceptar el tipo de datos XML en **sp_preparedocument,** y para generar XML y nuevas columnas de tipo SQL del conjunto de filas.

4.2 PROCESAMIENTO XML EN SQL SERVER

SQL Server permite proporcionar una base de datos que puede almacenar datos relacionales y XML.

4.2.1 Tipo de datos XML

4.2.1.1 XML sin tipo

El tipo de datos XML en SQL Server implementa la norma ISO estándar SQL-2003 tipo de datos XML. Como tal, puede almacenar no sólo bien formado XML 1.0 documentos, sino también fragmentos de XML llamado contento con los nodos de texto y un número arbitrario de elementos de nivel superior.

Comprueba la buena formación de los datos se lleva a cabo, que no requiere el tipo de datos XML en obligarse a los esquemas XML. Datos que no están bien formados son rechazados. XML sin tipo es útil cuando el esquema no se sabe *a priori* para que una solución basada en la cartografía no sea posible. También, es útil cuando el esquema se conoce, pero la asignación a un modelo de datos relacional es muy complejo y difícil de mantener, o existen varios esquemas y están obligados a fines de los datos basados en los requerimientos externos.

La siguiente sentencia crea una tabla llamada SOLICITUD con una clave entera y una columna XML sin tipo llamada detalle

```
USE NEGOCIOS2011
GO

CREATE TABLE CREDITO(
      ID INT PRIMARY KEY,
      DETALLE XML NOT NULL)
```

4.2.1.2 XML con tipo

Si se han definido esquemas XML que describe los datos XML, se puede asociar la colección de esquemas XML con la columna de XML para obtener XML con *tipo.* Los esquemas XML se utilizan para validar los datos, realizar comprobaciones de tipo más preciso que el XML sin tipo durante la compilación de las declaraciones de

modificación y consulta de datos y optimizar el almacenamiento y el procesamiento de consultas.

Las columnas XML, los parámetros y las variables pueden almacenar documentos XML o contenido, que se puede especificar como una opción (documento o contenido, respectivamente, con un contenido por defecto) en el momento de la declaración. Además, debe proporcionar la colección de esquemas XML.

Especificar el documento si cada instancia XML tiene exactamente un elemento de nivel superior, de lo contrario, usar el contenido. El compilador de consultas utiliza el indicador DOCUMENTO en los controles de tipo SINGLETON para inferir elementos de nivel superior.

La siguiente sentencia crea una tabla llamada **XmlCredito** con un **documento** XML con tipo columna con **myCollection.** La columna XML se especifica también que aceptar fragmentos de XML, no sólo los documentos XML.

```
USE NEGOCIOS2011
GO

CREATE TABLE XMLCREDITO (
      ID INT PRIMARY KEY,
      SOLICITUD XML(CONTENT MYCOLLECTION))
```

4.2.2 Almacenamiento de datos XML

Puede proporcionar un valor para una columna XML a través de un parámetro o una variable de varias maneras:

- Como tipo binario o de SQL que se convierte implícitamente en el tipo de datos XML.
- A través del contenido de un archivo.
- Como la salida del mecanismo de publicación XML FOR XML con la directiva TYPE, que genera una instancia de datos de tipo XML.

El valor proporcionado se comprueba la buena formación. Una columna XML por defecto permite a ambos documentos y fragmentos XML para ser almacenados. Si los

datos no pasan la comprobación de buena formación, se rechaza con un mensaje de error apropiado.

Para un dato XML con tipo, el valor proporcionado se comprueba la conformidad con los esquemas XML que se registran en la colección de esquemas XML que está escribiendo la columna XML. La instancia XML se rechaza si no está validación. Además, la bandera DOCUMENTO en XML restringe los valores aceptados para documentos XML sólo mientras que el contenido permite a ambos documentos XML y contenido para su entrega.

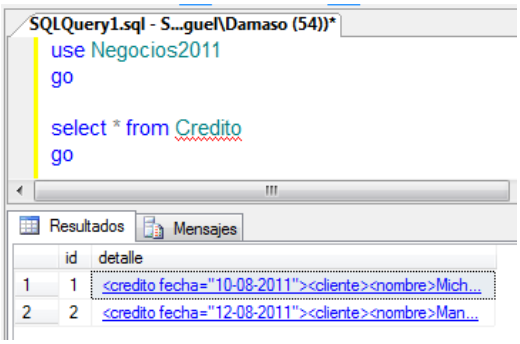
4.2.2.1 Insertar datos en una columna XML sin tipo

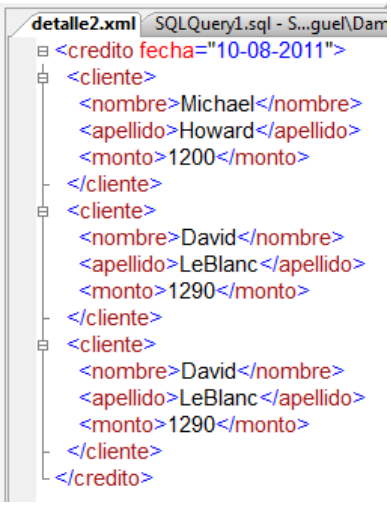
La siguiente declaración se inserta una nueva fila en la tabla denominada **CREDITOS** con el valor **1** para la columna de clave primaria y una instancia de **<credito>** para la columna XML. Los datos **<credito>,** suministrado como una cadena, se convierte implícitamente en el tipo de datos XML y comprobar la buena formación durante la inserción.

```
USE NEGOCIOS2011
GO

INSERT INTO CREDITO VALUES
(1,
'<CREDITO FECHA="10-08-2011">
      <CLIENTE>
            <NOMBRE>MICHAEL</NOMBRE>
            <APELLIDO>HOWARD</APELLIDO>
            <MONTO>1200</MONTO>
      </CLIENTE>
      <CLIENTE>
            <NOMBRE>DAVID</NOMBRE>
            <APELLIDO>LEBLANC</APELLIDO>
            <MONTO>1290</MONTO>
      </CLIENTE>
      <CLIENTE>
            <NOMBRE>DAVID</NOMBRE>
            <APELLIDO>LEBLANC</APELLIDO>
            <MONTO>1290</MONTO>
      </CLIENTE>
```

```
</CREDITO>'),
(2,
'<CREDITO FECHA="12-08-2011">
      <CLIENTE>
            <NOMBRE>MANUEL</NOMBRE>
            <APELLIDO>LUNA</APELLIDO>
            <MONTO>3400</MONTO>
      </CLIENTE>
      <CLIENTE>
            <NOMBRE>JORGE</NOMBRE>
            <APELLIDO>LAURA</APELLIDO>
            <MONTO>1290</MONTO>
      </CLIENTE>
      <CLIENTE>
            <NOMBRE>INES</NOMBRE>
            <APELLIDO>RIOS</APELLIDO>
            <MONTO>4500</MONTO>
      </CLIENTE>
</CREDITO>')
```

Para listar los registros de la tabla, ejecutamos la sintaxis siguiente, donde el proceso listará los registros y la estructura XML insertado. Para abrir un archivo, hacer un click al link, tal como se muestra.





Más adelante aprenderemos a consultar los datos de una estructura XML.

4.2.2.2 Insertar datos en la columna XML a través de un archivo de tipo XML

La instrucción INSERT en el segmento de código siguiente lee el contenido del archivo **C:\temp\credito.xml** como un BLOB mediante el uso de OPENROWSET. Una nueva fila se insertará en la tabla denominada **CREDITOS** con un valor de **3** para la clave principal y el contenido del archivo en la columna XML llamada **detalle**.

```
USE NEGOCIOS2011
GO

INSERT INTO CREDITO
SELECT 3, DETALLE
FROM (SELECT *
      FROM OPENROWSET
            (BULK 'C:\TEMP\CREDITOS.XML',
            SINGLE_BLOB)
      AS DETALLE) AS R(DETALLE)
GO
```

4.2.3 Recuperando datos de tipo XML

Una consulta SELECT devuelve los resultados como un conjunto de filas. Opcionalmente, se pueden recuperar resultados formales de una consulta SQL como XML especificando la cláusula FOR XML en la consulta. La cláusula FOR XML puede usarse en consultas de nivel superior y en subconsultas.

A continuación se muestra la sintaxis básica de la cláusula FOR XML

```
[ FOR { BROWSE | <XML> } ]
<XML> ::=
XML
    {
      { RAW [ ('ElementName') ] | AUTO }
        [
           <CommonDirectives>
           [ , { XMLDATA | XMLSCHEMA [ ('TargetNameSpaceURI') ]} ]
           [ , ELEMENTS [ XSINIL | ABSENT ]
        ]
      | EXPLICIT
        [
           <CommonDirectives>
           [ , XMLDATA ]
        ]
      | PATH [ ('ElementName') ]
```

```
        [
           <CommonDirectives>
           [ , ELEMENTS [ XSINIL | ABSENT ] ]
        ]
    }
<CommonDirectives> ::=
  [ , BINARY BASE64 ]
  [ , TYPE ]
  [ , ROOT [ ('RootName') ] ]
```

Argumentos	**Descripción**
RAW[('*ElementName*')]	Obtiene el resultado de la consulta y transforma cada fila del conjunto de resultados en un elemento XML con un identificador genérico <row /> como etiqueta del elemento. Opcionalmente, puede especificar un nombre para el elemento de fila cuando se utiliza esta directiva. El XML resultante utilizará el *ElementName* especificado como el elemento de fila generado para cada fila.
AUTO	Devuelve los resultados de la consulta en un árbol anidado XML sencillo. Cada tabla en la cláusula FROM de la que al menos se presenta una columna en la cláusula SELECT se representa como un elemento XML. A las columnas presentadas en la cláusula SELECT se les asignan los atributos de elemento apropiados.
EXPLICIT	Especifica que la forma del árbol XML resultante está definida explícitamente. Con este modo, es necesario escribir las consultas de una cierta manera, de modo que se pueda especificar explícitamente información adicional acerca de la anidación deseada.
PATH	Facilita la mezcla de elementos y atributos, así como la especificación de anidación adicional para representar propiedades complejas. Puede utilizar consultas FOR XML de modo EXPLICIT para construir XML a partir de un conjunto de filas, pero el modo PATH supone una alternativa más simple a las consultas de modo EXPLICIT potencialmente complicadas. De forma predeterminada, el modo PATH genera un contenedor

	de elementos <row> para cada fila del conjunto de resultados. También se puede especificar un nombre de elemento. En este caso, el nombre especificado se utilizará como nombre del elemento contenedor. Si se proporciona una cadena vacía (FOR XML PATH ('')), no se generará ningún elemento contenedor.
XMLDATA	Especifica que se debe devolver un esquema XDR (XML-Data Reduced) insertado. El esquema se antepone al documento como un esquema insertado.
XMLSCHEMA	Devuelve un esquema XML W3C (XSD) insertado. Opcionalmente, puede especificar un URI de espacio de nombres de destino al especificar esta directiva. De este modo, se devuelve el espacio de nombres especificado en el esquema.
ELEMENTS	Si se especifica la opción ELEMENTS, las columnas se devuelven como subelementos. Sin embargo, se les asignan atributos XML. Esta opción solo se admite en los modos RAW, AUTO y PATH. También puede especificar XSINIL o ABSENT cuando utilice esta directiva. XSINIL especifica que se puede crear un elemento con un atributo **XSI:NIL** establecido en True para los valores de columna NULL. De forma predeterminada, o cuando se especifica ABSENT junto con ELEMENTS, no se crea ningún elemento para los valores NULL.
BINARY BASE64	Si se especifica la opción BINARY Base64, todos los datos binarios que devuelve la consulta se representan en formato codificado base64. Para recuperar datos binarios mediante el modo RAW y EXPLICIT, se debe especificar esta opción. En modo AUTO, de forma predeterminada, se devuelven datos binarios como una referencia.
TYPE	Especifica que la consulta devuelve los resultados como el tipo **XML**.
ROOT [('*RootName*')]	Especifica que se puede agregar un solo elemento de

	nivel superior al XML resultante. También se puede especificar el nombre del elemento raíz que se generará. El valor predeterminado es "root".

La cláusula FOR XML de nivel superior sólo puede usarse en la instrucción SELECT. En el caso de las subconsultas, FOR XML puede usarse en las instrucciones INSERT, UPDATE y DELETE. También puede usarse en instrucciones de asignación.

En una cláusula FOR XML se especifica uno de estos modos:

- RAW
- AUTO
- EXPLICIT
- PATH

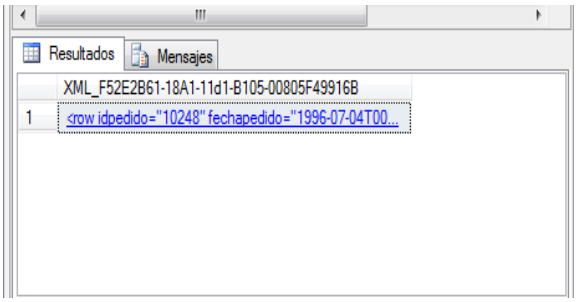
4.2.3.1 Usar Modo RAW

El modo RAW transforma cada fila del conjunto de resultados de la consulta en un elemento XML que tiene el identificador genérico <row> o el nombre del elemento, que se proporciona de manera opcional. De manera predeterminada, cada valor de columna del conjunto de filas que no es NULL se asigna a un atributo del elemento <row>. Si se agrega la directiva ELEMENTS a la cláusula FOR XML, cada valor de columna se asigna a un subelemento del elemento <row>. Opcionalmente, junto con la directiva ELEMENTS se puede especificar la opción XSINIL para asignar los valores de columna NULL del conjunto de resultados a un elemento que tenga el atributo, xsi:nil="true".

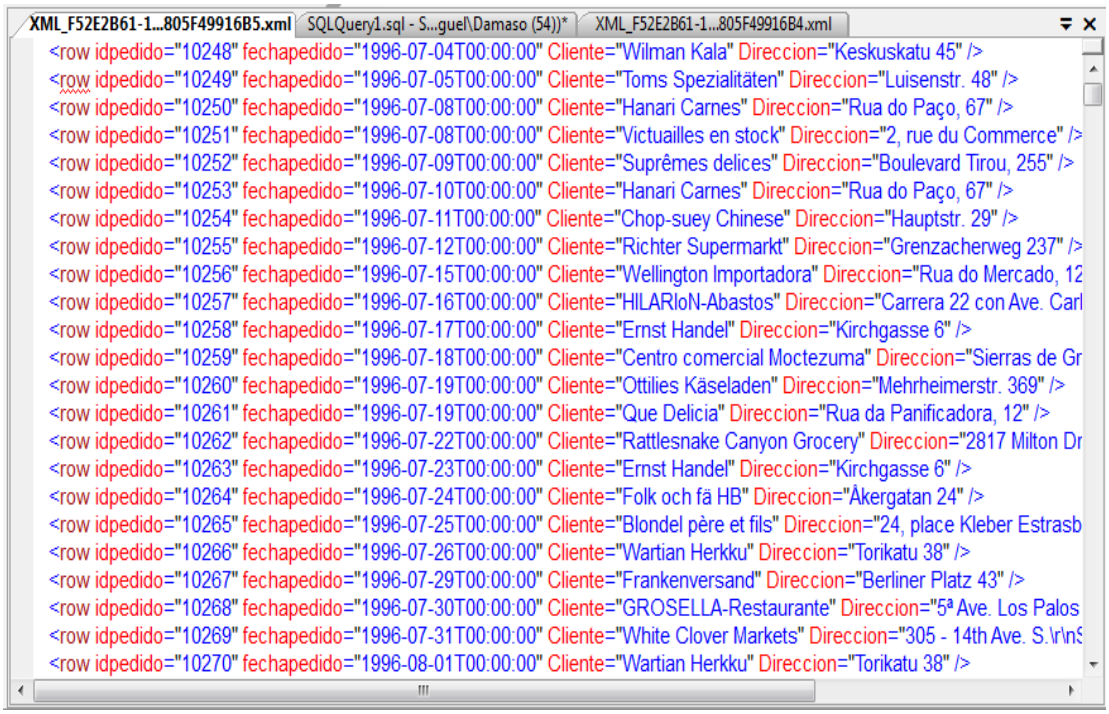
El siguiente ejemplo nos permite recuperar un fragmento XML que contienen los datos utilizando la cláusula FOR XML en modo RAW

```
USE NEGOCIOS2011
GO

SELECT IDPEDIDO,
       FECHAPEDIDO,
       C.NOMCLIENTE AS CLIENTE,
       C.DIRCLIENTE AS DIRECCION
```

```
FROM VENTAS.CLIENTES C JOIN VENTAS.PEDIDOSCABE P
ON C.IDCLIENTE = P.IDCLIENTE
ORDER BY IDPEDIDO
FOR XML RAW
GO
```



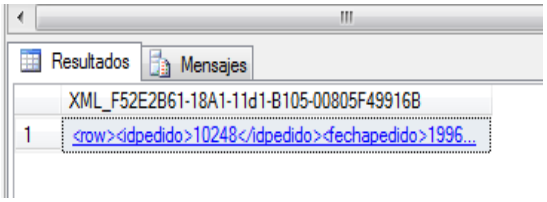
La consulta produce un fragmento XML que contiene elementos <row>. Al abrir el fragmento se lista todas las filas de la consulta en formato XML, tal como se muestra en la figura siguiente



El siguiente ejemplo muestra cómo recuperar la data como elementos en lugar de atributos especificando la opción ELEMENTS

```
USE NEGOCIOS2011
GO
```

```
SELECT IDPEDIDO,
       FECHAPEDIDO,
       C.NOMCLIENTE AS CLIENTE,
       C.DIRCLIENTE AS DIRECCION
FROM VENTAS.CLIENTES C JOIN VENTAS.PEDIDOSCABE P
ON C.IDCLIENTE = P.IDCLIENTE
ORDER BY IDPEDIDO
FOR XML RAW, ELEMENTS
GO
```



La consulta produce el fragmento XML en el formato que se muestra.

El formato XML, los campos están distribuidos como elementos de <row>.

4.2.3.2 Usar Modo AUTO

El modo AUTO devuelve los resultados de la consulta como elementos XML anidados. Esto no ofrece un gran control sobre la forma del XML generado a partir del resultado de una consulta. Las consultas en modo AUTO son útiles si desea generar jerarquías sencillas. Sin embargo, Usar el modo EXPLICIT y Usar el modo PATH ofrecen mayor

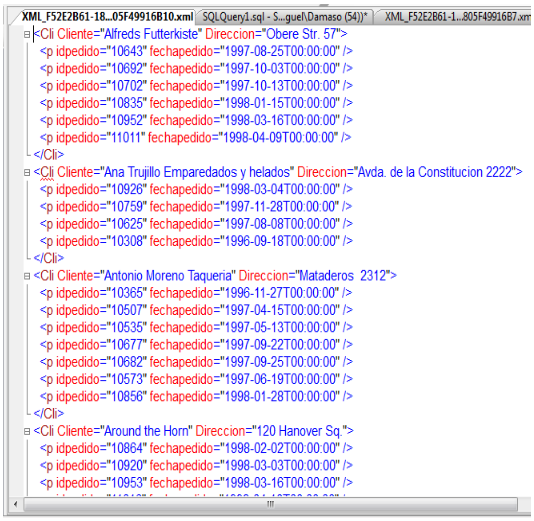
control y flexibilidad a la hora de decidir la forma del XML procedente del resultado de una consulta.

Cada tabla de la cláusula FROM, de la que al menos se presenta una columna en la cláusula SELECT, se representa como un elemento XML. Las columnas que se incluyen en la cláusula SELECT se asignan a atributos o subelementos, si se especifica la opción ELEMENTS en la cláusula FOR XML.

La jerarquía XML, anidamiento de los elementos, del XML resultante está basada en el orden de las tablas identificadas por las columnas especificadas en la cláusula SELECT. Por tanto, el orden en que se especifican los nombres de columna en la cláusula SELECT es importante. La primera, la tabla situada más a la izquierda que se identifica, constituye el elemento superior del documento XML resultante. La segunda tabla situada más a la izquierda, identificada por las columnas de la instrucción SELECT, constituye un subelemento del elemento superior, etc.

En el siguiente ejemplo, muestra cómo utilizar la consulta de modo AUTO que retorne un fragmento XML que contiene la lista de pedidos por cliente

```
SELECT C.NOMCLIENTE AS CLIENTE,
       C.DIRCLIENTE AS DIRECCION,
       IDPEDIDO,
       FECHAPEDIDO
FROM VENTAS.CLIENTES C JOIN VENTAS.PEDIDOSCABE P
ON C.IDCLIENTE = P.IDCLIENTE
ORDER BY NOMCLIENTE
FOR XML AUTO
GO
```



```xml
<Cli Cliente="Alfreds Futterkiste" Direccion="Obere Str. 57">
  <p idpedido="10643" fechapedido="1997-08-25T00:00:00" />
  <p idpedido="10692" fechapedido="1997-10-03T00:00:00" />
  <p idpedido="10702" fechapedido="1997-10-13T00:00:00" />
  <p idpedido="10835" fechapedido="1998-01-15T00:00:00" />
  <p idpedido="10952" fechapedido="1998-03-16T00:00:00" />
  <p idpedido="11011" fechapedido="1998-04-09T00:00:00" />
</Cli>
<Cli Cliente="Ana Trujillo Emparedados y helados" Direccion="Avda. de la Constitucion 2222">
  <p idpedido="10926" fechapedido="1998-03-04T00:00:00" />
  <p idpedido="10759" fechapedido="1997-11-28T00:00:00" />
  <p idpedido="10625" fechapedido="1997-08-08T00:00:00" />
  <p idpedido="10308" fechapedido="1996-09-18T00:00:00" />
</Cli>
<Cli Cliente="Antonio Moreno Taqueria" Direccion="Mataderos  2312">
  <p idpedido="10365" fechapedido="1996-11-27T00:00:00" />
  <p idpedido="10507" fechapedido="1997-04-15T00:00:00" />
  <p idpedido="10535" fechapedido="1997-05-13T00:00:00" />
  <p idpedido="10677" fechapedido="1997-09-22T00:00:00" />
  <p idpedido="10682" fechapedido="1997-09-25T00:00:00" />
  <p idpedido="10573" fechapedido="1997-06-19T00:00:00" />
  <p idpedido="10856" fechapedido="1998-01-28T00:00:00" />
</Cli>
<Cli Cliente="Around the Horn" Direccion="120 Hanover Sq.">
  <p idpedido="10864" fechapedido="1998-02-02T00:00:00" />
  <p idpedido="10920" fechapedido="1998-03-03T00:00:00" />
  <p idpedido="10953" fechapedido="1998-03-16T00:00:00" />
```

En el fragmento XML, agrupa los pedidos por cada cliente, donde los atributos representan el nombre del cliente y su dirección; sus elementos son los datos de cada pedido.

En el siguiente ejemplo, vamos a recuperar los datos como elementos en lugar de atributos especificando con la opción ELEMENTS.

```sql
SELECT C.NOMCLIENTE AS CLIENTE,
       C.DIRCLIENTE AS DIRECCION,
       IDPEDIDO,
       FECHAPEDIDO
FROM VENTAS.CLIENTES C JOIN VENTAS.PEDIDOSCABE P
ON C.IDCLIENTE = P.IDCLIENTE
ORDER BY NOMCLIENTE
FOR XML AUTO, ELEMENTS
GO
```

```xml
<Cli>
  <Cliente>Alfreds Futterkiste</Cliente>
  <Direccion>Obere Str. 57</Direccion>
  <p>
    <idpedido>10643</idpedido>
    <fechapedido>1997-08-25T00:00:00</fechapedido>
  </p>
  <p>
    <idpedido>10692</idpedido>
    <fechapedido>1997-10-03T00:00:00</fechapedido>
  </p>
  <p>
    <idpedido>10702</idpedido>
    <fechapedido>1997-10-13T00:00:00</fechapedido>
  </p>
  <p>
    <idpedido>10835</idpedido>
    <fechapedido>1998-01-15T00:00:00</fechapedido>
  </p>
  <p>
    <idpedido>10952</idpedido>
    <fechapedido>1998-03-16T00:00:00</fechapedido>
  </p>
  <p>
    <idpedido>11011</idpedido>
    <fechapedido>1998-04-09T00:00:00</fechapedido>
  </p>
```

El fragmento XML, visualizamos los datos como elementos, donde agrupamos por Cliente, y por cada cliente listamos los pedidos.

4.2.3.3 Usar Modo EXPLICIT

El modo EXPLICIT concede un mayor control de la forma del XML. Es posible mezclar atributos y elementos con total libertad para decidir la forma del XML. Requiere un formato específico para el conjunto de filas resultante generado debido a la ejecución de la consulta. Después, el formato del conjunto de filas se asigna a una forma de XML.

El modo EXPLICIT transforma en un documento XML el conjunto de filas resultante de la ejecución de la consulta. Para que el modo EXPLICIT, pueda generar el documento XML, el conjunto de filas debe ajustarse a un determinado formato. Por ello, es necesario escribir la consulta SELECT para generar el conjunto de filas, la **tabla universal**, con un formato específico que permita a la lógica del procesamiento generar el XML deseado.

En primer lugar, la consulta debe crear las dos columnas de metadatos siguientes:

- La primera columna debe proporcionar el número de etiqueta, el tipo de entero, del elemento actual, y el nombre de la columna debe ser **Tag**. La consulta debe proporcionar un número de etiqueta único para cada elemento que se vaya a construir a partir del conjunto de filas.
- La segunda columna debe proporcionar un número de etiqueta del elemento primario, y el nombre de la columna debe ser **Parent**. De este modo, las columnas Tag y Parent ofrecen información sobre la jerarquía.

Los valores de estas columnas de metadatos, junto con la información de los nombres de columna, se usan para generar el XML deseado. Tenga en cuenta que la consulta debe proporcionar los nombres de columna de una manera determinada. Observe también que un valor 0 ó NULL en la columna **Parent** indica que el elemento correspondiente no tiene uno primario. El elemento se agrega al XML como elemento de nivel superior.

En el siguiente ejemplo, vamos a listar los datos de los clientes, donde el campo id y nombre serán atributos, y el campo dirección y teléfono serán elementos.

```
SELECT 1 AS TAG,
      NULL AS PARENT,
      IDCLIENTE AS [CLIENTE!1!CODIGO],
      NOMCLIENTE AS [CLIENTE!1!NOMBRE],
      DIRCLIENTE AS [CLIENTE!1!DIRECCION!ELEMENT],
      TELEFONO AS [CLIENTE!1!TELEFONO!ELEMENT]
FROM VENTAS.CLIENTES
FOR XML EXPLICIT
GO
```

El fragmento XML lista las filas de la tabla de clientes, donde el código y el nombre son atributos; y los campos dirección y teléfono son elementos.

4.2.3.4 Usar Modo PATH

El modo PATH facilita la combinación de elementos y atributos. También, facilita la especificación de anidación adicional para representar propiedades complejas. Puede utilizar consultas de modo FOR XML EXPLICIT para generar XML a partir de un conjunto de filas, pero el modo PATH supone una alternativa más sencilla a las consultas de modo EXPLICIT potencialmente complicadas. El modo PATH, junto con la posibilidad de escribir consultas FOR XML anidadas y la directiva TYPE para devolver instancias de tipo **XML**, permite escribir consultas de forma más fácil.

En el modo PATH, los nombres o alias de columna se tratan como expresiones XPath. Estas expresiones indican el modo en el que se asignan los valores a XML. Cada expresión XPath es una expresión relativa que proporciona el tipo de elemento, como el atributo, el elemento y el valor escalar, así como el nombre y la jerarquía del nodo que se generará en relación con el elemento de fila.

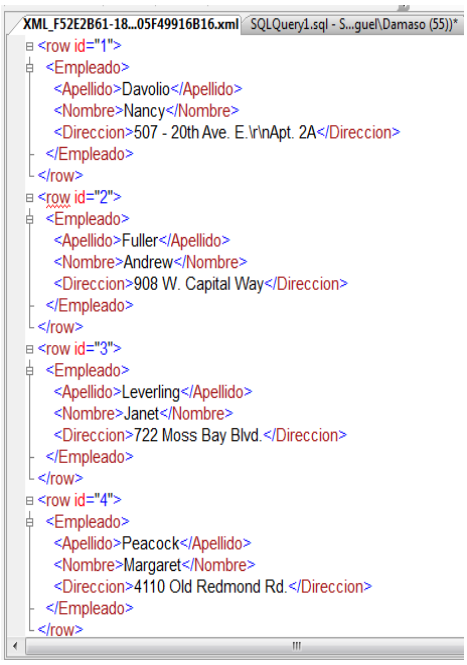
En el siguiente ejemplo, vamos a ejecutar una consulta de pedidos cuyo año de pedido sea en el 2011 en modo FOR XML PATH.

```
SELECT IDPEDIDO, FECHAPEDIDO, DESTINATARIO
FROM VENTAS.PEDIDOSCABE
WHERE DATEPART(YY,FECHAPEDIDO)=2011
FOR XML PATH
GO
```

El resultado siguiente es XML centrado en elementos en el que cada valor de columna del conjunto de filas resultante se agrupa en un elemento. Puesto que la cláusula SELECT no especifica ningún alias para los nombres de columna, los nombres de elemento secundario generados son los mismos que los nombres de columna correspondientes de la cláusula SELECT. Para cada fila del conjunto de filas, se agrega una etiqueta <row>.

El siguiente ejemplo muestra cómo utilizar la consulta de modo PATH y retorna un fragmento XML que contiene la lista de los empleados

```
SELECT IDEMPLEADO "@ID",
      APELLIDOS "EMPLEADO/APELLIDO",
      NOMBRE "EMPLEADO/NOMBRE",
      DIRECCION "EMPLEADO/DIRECCION"
FROM RRHH.EMPLEADOS
FOR XML PATH
GO
```



El id del empleado es mapeado como atributo con el signo (@), las columnas apellidos, nombre y dirección están mapeados como sub elementos de “Empleado” con la marca slash “/”.

El siguiente ejemplo muestra cómo utilizar el argumento opcional ElementName a la consulta de modo PATH para modificar el nombre del elemento row.

```
SELECT IDEMPLEADO "@ID",
      APELLIDOS "EMPLEADO/APELLIDO",
      NOMBRE "EMPLEADO/NOMBRE",
      DIRECCION "EMPLEADO/DIRECCION"
FROM TB_EMPLEADOS
FOR XML PATH('EMPLOYEE')
GO
```

4.2.4 Recuperar datos con OPENXML

OPENXML, palabra clave de Transact-SQL, proporciona un conjunto de filas en documentos XML en memoria que es similar a una tabla o una vista. OPENXML permite el acceso a los datos XML a pesar de ser un conjunto de filas relacional. Para ello, proporciona una vista de conjunto de filas de la representación interna de un documento XML. Los registros del conjunto de filas pueden almacenarse en tablas de base de datos.

OPENXML puede utilizarse en instrucciones SELECT y SELECT INTO donde puedan aparecer como origen proveedores de conjuntos de filas, una vista u OPENROWSET.

Para escribir consultas en un documento XML mediante OPENXML, primero es necesario llamar a **sp_xml_preparedocument**, que analiza el documento XML y devuelve un identificador para el documento analizado que está listo para su uso. El documento analizado es una representación en árbol del modelo de objetos de documento (DOM) de los distintos nodos del documento XML. Este identificador de documento se pasa a OPENXML. A continuación, OPENXML proporciona una vista de conjunto de filas del documento, basándose en los parámetros que ha recibido.

4.2.4.1 Ejecutar una instrucción SELECT simple con OPENXML

El documento XML de este ejemplo se compone de los elementos <Clientes>, <Pedido> y <PedidoDetalle>. La instrucción OPENXML recupera la información del cliente desde el documento XML en un conjunto de filas de dos columnas (**IDCliente** y **Nombre**).

Primero, para obtener un identificador de documento se llama al procedimiento almacenado **sp_xml_preparedocument**. Este identificador de documento se pasa a OPENXML.

La instrucción OPENXML muestra lo siguiente:

- *rowpattern* (/ROOT/Clientes) identifica los nodos <Clientes> que se van a procesar.
- El valor del parámetro *flags* se establece en **1** e indica una asignación centrada en atributos. Como resultado, los atributos XML se asignan a las columnas del conjunto de filas definido en *SchemaDeclaration*.

- En *SchemaDeclaration*, en la cláusula WITH, los valores *ColName* especificados coinciden con los nombres de atributo XML correspondientes. Por lo tanto, el parámetro *ColPattern* no se especifica en *SchemaDeclaration*.

A continuación, la instrucción SELECT recupera todas las columnas del conjunto de filas que proporciona OPENXML.

```
DECLARE @DocHandle int
DECLARE @XmlDocument nvarchar(1000)
SET @XmlDocument = N'<ROOT>
<Cliente IDCliente="VINET" Nombre="Paul Henriot">
   <Pedido IDpedido="10248" IDCliente="VINET" IDEmpleado="5"
          Fecha="1996-07-04T00:00:00">
      <PedidoDetalle IDProducto="11" Cantidad="12"/>
      <PedidoDetalle IDProducto="42" Cantidad="10"/>
   </Pedido>
</Cliente>
<Cliente IDCliente="LILAS" Nombre="Carlos Gonzlez">
   <Pedido IDpedido="10283" IDCliente="LILAS" IDEmpleado="3"
          Fecha="1996-08-16T00:00:00">
      <PedidoDetalle IDProducto="72" Cantidad="3"/>
   </Pedido>
</Cliente>
</ROOT>'
-- Crear la representacion interna del documento XML.
EXEC sp_xml_preparedocument @DocHandle OUTPUT, @XmlDocument
-- Ejecutar una sentencia SELECT utilizando OPENXML .
SELECT *
FROM OPENXML (@DocHandle, '/ROOT/Cliente',1)
      WITH (IDCliente  varchar(10),
            Nombre varchar(20))
EXEC sp_xml_removedocument @DocHandle
Go
```

4.2.4.2 Especificar ColPattern para la asignación entre columnas del conjunto de filas y los atributos y elementos XML

Este ejemplo muestra cómo se especifica el patrón XPath en el parámetro opcional *ColPattern* para proporcionar una asignación entre columnas del conjunto de filas y los atributos y elementos XML. El documento XML de este ejemplo se compone de los

elementos <Cliente>, <Pedido> y <PedidoDetalle>. La instrucción OPENXML recupera la información del cliente y del pedido del documento XML en forma de conjunto de filas (**IDCliente**, **Fecha**, **IDProducto** y **Cantidad**).

Primero, para obtener un identificador de documento se llama al procedimiento almacenado **sp_xml_preparedocument**. Este identificador de documento se pasa a OPENXML. La instrucción OPENXML muestra lo siguiente:

- *rowpattern* (/ROOT/Cliente/Pedido/PedidoDetalle) identifica los nodos <PedidoDetalle> que se van a procesar.

A modo de ilustración, el valor del parámetro *flags* se establece en **2** e indica una asignación centrada en elementos. Sin embargo, la asignación especificada en *ColPattern* sobrescribe esta asignación,es decir, el patrón XPath especificado en *ColPattern* asigna a atributos las columnas del conjunto de filas. El resultado es una asignación centrada en atributos.

En *SchemaDeclaration*, en la cláusula WITH, también se especifica *ColPattern* con los parámetros *ColName* y *ColType*. El parámetro opcional *ColPattern* es el patrón XPath especificado e indica lo siguiente:

- Las columnas **IDPedido**, **IDCliente** y **Fechae** del conjunto de filas se asignan a los atributos del elemento primario de los nodos identificados por *rowpattern*, y *rowpattern* identifica los nodos <OrderDetail>. Por lo tanto, las columnas **IDCliente** y **Fecha** se asignan a los atributos **IDCliente** y **Fecha** del elemento <Order>.
- Las columnas **IDProducto** y **Cantidad** del conjunto de filas se asignan a los atributos **IDProducto** y **Cantidad** de los nodos identificados en *rowpattern*.

A continuación, la instrucción SELECT recupera todas las columnas del conjunto de filas que proporciona OPENXML.

```
DECLARE @XmlDocumentHandle int
DECLARE @XmlDocument nvarchar(1000)
SET @XmlDocument = N'<ROOT>
<Cliente IDCliente="VINET" Nombre="Paul Henriot">
   <Pedido IDpedido="10248" IDCliente="VINET" IDEmpleado="5"
          Fecha="1996-07-04T00:00:00">
      <PedidoDetalle IDProducto="11" Cantidad="12"/>
      <PedidoDetalle IDProducto="42" Cantidad="10"/>
   </Pedido>
</Cliente>
<Cliente IDCliente="LILAS" Nombre="Carlos Gonzlez">
   <Pedido IDpedido="10283" IDCliente="LILAS" IDEmpleado="3"
          Fecha="1996-08-16T00:00:00">
      <PedidoDetalle IDProducto="72" Cantidad="3"/>
   </Pedido>
</Cliente>
</ROOT>'

-- Crea una representacion interna del documento XML.
EXEC sp_xml_preparedocument @XmlDocumentHandle OUTPUT, @XmlDocument

-- Ejecuta la sentencia SELECT utilziando OPENXML
SELECT * FROM OPENXML (@XmlDocumentHandle,
'/ROOT/Cliente/Pedido/PedidoDetalle',2)
WITH (IDpedido    int         '../@IDpedido',
      IDCliente   varchar(10) '../@IDCliente',
      Fecha       datetime    '../@Fecha',
      IDProducto  int         '@IDProducto',
      Cantidad    int         '@Cantidad')

EXEC sp_xml_removedocument @XmlDocumentHandle
```

Resumen

- Los datos XML pueden interactuar con datos relacionales y aplicaciones SQL. Esto significa que XML puede ser introducido en el sistema según las necesidades de modelado de datos se presentan sin interrumpir las aplicaciones existentes. El servidor de base de datos también proporciona una funcionalidad administrativa para la gestión de datos XML (por ejemplo, BACKUP, recuperación y replicación).
- SQL Server 2008 introduce un tipo de datos nativo llamado XML. Un usuario puede crear una tabla que tiene una o más columnas de tipo XML, además de columnas relacionales. Las variables y los parámetros de XML también son permitidos.
- Puede crear una tabla con una columna XML mediante la instrucción CREATE TABLE de costumbre. La columna XML puede ser indexado de una manera especial. Puede trabaja con formato o sin formato
- Para almacenar datos XML, puede proporcionar un valor para una columna XML a través de un parámetro o una variable de varias maneras:
  - Como tipo binario o de SQL que se convierte implícitamente en el tipo de datos XML.
  - A través del contenido de un archivo.
  - Como la salida del mecanismo de publicación XML FOR XML con la directiva TYPE, que genera una instancia de datos de tipo XML.
- El valor proporcionado se comprueba la buena formación. Una columna XML por defecto permite a ambos documentos y fragmentos XML para ser almacenados. Si los datos no pasan la comprobación de buena formación, se rechaza con un mensaje de error apropiado.
- Una consulta SELECT devuelve los resultados como un conjunto de filas. Opcionalmente, se pueden recuperar resultados formales de una consulta SQL como XML especificando la cláusula FOR XML en la consulta. La cláusula FOR XML puede usarse en consultas de nivel superior y en subconsultas.
- El modo RAW transforma cada fila del conjunto de resultados de la consulta en un elemento XML que tiene el identificador genérico <row> o el nombre del elemento, que se proporciona de manera opcional. El modo AUTO devuelve los resultados de la consulta como elementos XML anidados. El modo EXPLICIT concede un mayor

control de la forma del XML. Es posible mezclar atributos y elementos con total libertad para decidir la forma del XML.

📖 OPENXML, palabra clave de Transact-SQL, proporciona un conjunto de filas en documentos XML en memoria que es similar a una tabla o una vista. OPENXML permite el acceso a los datos XML a pesar de ser un conjunto de filas relacional.

📖 Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

- http://technet.microsoft.com/es-es/library/ms187897(SQL.90).aspx

  Aquí hallará los conceptos de OPENXML

- http://msdn.microsoft.com/es-es/library/ms178107.aspx

  En esta página, hallará los conceptos de consulta FOR XML

- http://msdn.microsoft.com/en-us/library/ms345117(v=sql.90).aspx

  Aquí hallará los conceptos de manejo de datos en formato XML

UNIDAD DE APRENDIZAJE
5

MANEJO DE USUARIOS EN SQL SERVER

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno, haciendo uso de los comandos del SQL SERVER o utilizando asistentes, crearán un inicio de sesión autenticado en SQL SERVER que otorgará o denegará permisos sobre los objetos de una base de datos.

TEMARIO

1.1. Introducción
1.1.1. Entidades de seguridad
1.1.2. Autenticación
1.1.3. Inicio de sesión y usuarios
1.1.3.1. Creación de login
1.1.3.2. Creación de usuarios en SQL Server
1.1.3.3. Manejo de roles a nivel servidor
1.1.3.4. Manejo de roles a nivel base de datos
1.1.4. Permisos en el motor de base de datos
1.1.4.1. Comando GRANT
1.1.4.2. Comando DENY
1.1.4.3. Comando REVOKE

ACTIVIDADES PROPUESTAS

- Los alumnos crea usuarios para manejar los objetos de la base de datos.
- Los alumnos asigna permisos a los usuarios de una base de datos para realizar operaciones de manipulación de datos y definición de datos.

5.1 INTRODUCCION

SQL Server 2008 es compatible con los modos de autenticación de Windows y se mezcla y se integra estrechamente con él. En este modo de acceso, se concede sobre la base de un TOKEN de seguridad asignada durante el inicio de sesión de dominio con éxito por una cuenta de Windows y SQL Server se solicita el acceso posteriormente. La condición previa es que ambos deben pertenecer al mismo entorno de ventanas.

El entorno de dominio de ACTIVE DIRECTORY proporciona un nivel adicional de protección del protocolo KERBEROS. Este protocolo regula el comportamiento del mecanismo de autenticación de Windows. En el servidor de autenticación de modo mixto SQL, también se puede utilizar. Las credenciales se verifican desde el repositorio mantenido por el servidor SQL Server.

El aumento de la seguridad se ha despedido de la necesidad de mantener el conjunto separado de cuentas. Sin embargo, los inicios de sesión de SQL Server se han mejorado con el cifrado de los Certificados de SQL generado para las comunicaciones que involucran el software de cliente basado en MADC. NET.

SQL Server incluye varios métodos y herramientas que permiten configurar la seguridad para que los usuarios, los servicios y las otras cuentas puedan tener acceso al sistema.

5.1.1 Entidades de seguridad

Las *entidades de seguridad* son entidades que pueden solicitar recursos de SQL Server. Igual que otros componentes del modelo de autorización de SQL Server, las entidades de seguridad se pueden organizar en jerarquías.

El ámbito de influencia de una entidad de seguridad depende del ámbito de su definición: Windows, servidor o base de datos; y de si la entidad de seguridad es indivisible o es una colección. Un Inicio de sesión de Windows es un ejemplo de entidad de seguridad indivisible y un Grupo de Windows es un ejemplo de una del tipo colección. Toda entidad de seguridad tiene un identificador de seguridad (SID).

Entidades de seguridad a nivel de Windows

- Inicio de sesión del dominio de Windows
- Inicio de sesión local de Windows

Entidad de seguridad de SQL Server

- Inicio de sesión de SQL Server

Entidades de seguridad a nivel de bases de datos

- Usuario de base de datos
- Función de base de datos
- Función de aplicación

5.1.1.1 Inicio de sesión sa de SQL Server

El inicio de sesión **sa** de SQL Server es una entidad de seguridad del servidor. Se crea de forma predeterminada cuando se instala una instancia. En SQL Server 2005 y SQL Server 2008, la base de datos predeterminada de **sa** es **master**. Es un cambio de comportamiento con respecto a versiones anteriores de SQL Server.

5.1.1.2 Función PUBLIC de base de datos

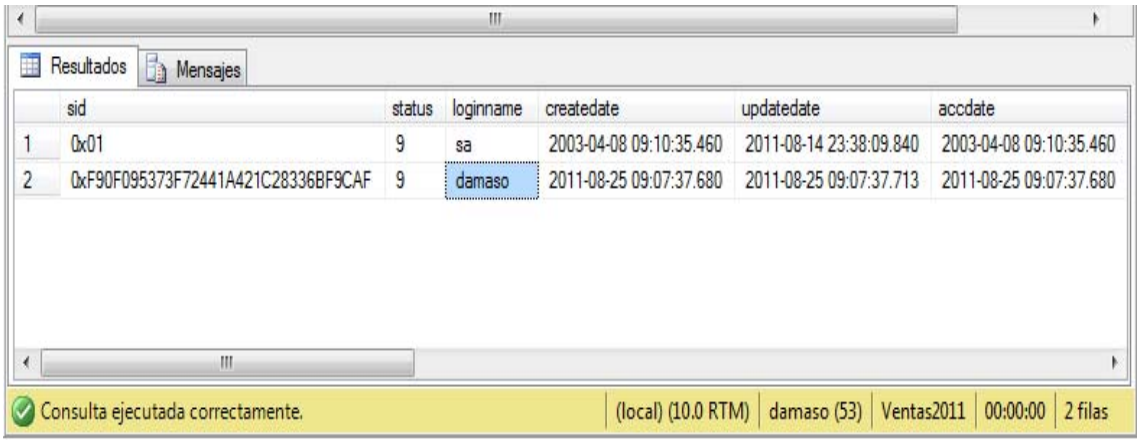
Todos los usuarios de una base de datos pertenecen a la función **public** de la base de datos. Cuando a un usuario no se le han concedido ni denegado permisos de un elemento que puede protegerse, el usuario hereda los permisos de ese elemento concedidos a **public**.

5.1.1.3 Cliente y servidor de base de datos

Por definición, un cliente y un servidor de base de datos son entidades de seguridad y se pueden proteger. Estas entidades se pueden autenticar mutuamente antes de establecer una conexión de red segura. SQL Server admite el protocolo de autenticación Kerberos, que define cómo interactúan los clientes con un servicio de autenticación de red.

En este ejemplo, listamos todos los usuarios de la base de datos Negocios2011, donde le resultado se muestra en la figura siguiente

```
USE NEGOCIOS2011
GO

SELECT *
FROM SYS.SYSLOGINS
GO
```



5.1.2 Autenticación

Para acceder a SQL SERVER, se dispone de dos modos de autenticación: Autenticación de Windows y Autenticación en modo mixto.

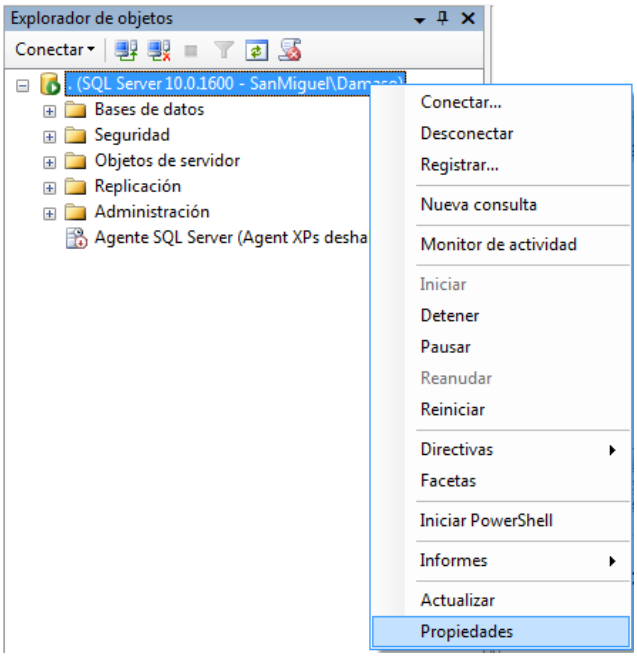
En la autenticación de Windows, el sistema operativo es el responsable de identificar al usuario.

SQL Server usa después esa identificación del sistema operativo para determinar los permisos del usuario que hay que aplicar. Con la autenticación en modo mixto, los dos Windows y SQL Server son responsables de identificar al usuario.

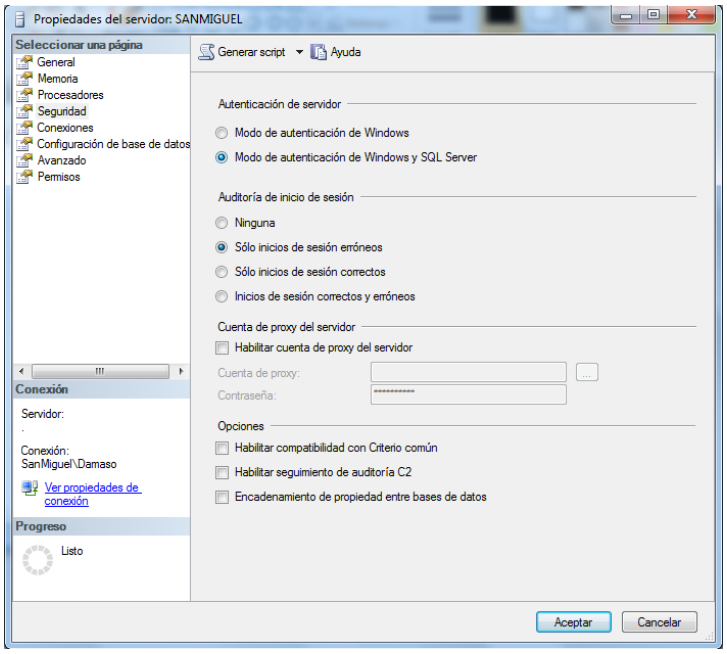
5.1.2.1 Configurar el modo de autenticación

En la configuración del modo de autenticación, se siguen los siguientes pasos:

En la ventana del explorador de objetos, hacer click derecho sobre el administrador del SQL Server, seleccione la opción Propiedades, tal como se muestra en la figura.



En la ventana de propiedades, selecciona la opción SEGURIDAD, para visualizar las opciones de seguridad. Seleccione el modo de autenticación de servidor, seleccione Modo Windows y SQL Server, modo mixto, para que el usuario ingrese por cualquier de las dos formas, cuando realice los cambios presione el botón ACEPTAR



5.1.3 Inicios de sesión y usuarios

En las siguientes secciones, aprenderemos cómo crear inicios de sesión y usuarios, pero antes de iniciar el proceso es necesario comprender que es un inicio de sesión.

Según lo visto anteriormente, una cuenta de usuario de Windows puede ser necesaria para conectarse a una base de datos. También, podría ser necesaria la autenticación de SQL Server. Tanto si se va a usar autenticación de Windows o autenticación de modo mixto, la cuenta que se usa para la conexión a SQL SERVER se le conoce como inicio de sesión de SQL Server.

En forma predeterminada, una cuenta de inicio de sesión de SQL Server no tiene un ID de usuario de base de datos asociado con ella, por ello carece de permisos.

5.1.3.1 Creación de Inicios de Sesión o Login

Podemos manejar un inicio de sesión utilizando el explorador de objetos en el manejador del SQL Server o ejecutando las sentencias CREATE LOGIN, ALTER LOGIN y DROP LOGIN

Para crear un inicio de sesión o login, usamos la sentencia CREATE LOGIN, la sintaxis es la siguiente:

```
CREATE LOGIN loginName { WITH <option_list1> | FROM <sources> }
<option_list1> ::=
    PASSWORD = { 'password' | hashed_password HASHED } [ MUST_CHANGE
]
    [ , <option_list2> [ ,... ] ]

<option_list2> ::=
    SID = sid
    | DEFAULT_DATABASE =database
    | DEFAULT_LANGUAGE =language
    | CHECK_EXPIRATION = { ON | OFF}
    | CHECK_POLICY = { ON | OFF}
    | CREDENTIAL =credential_name <sources> ::=
    WINDOWS [ WITH <windows_options>[ ,... ] ]
```

```
    | CERTIFICATE certname
    | ASYMMETRIC KEY asym_key_name<windows_options> ::=
    DEFAULT_DATABASE =database
    | DEFAULT_LANGUAGE =language
```

Argumentos	**Descripción**
loginName	Especifica el nombre del inicio de sesión que se va a crear. Hay cuatro tipos de inicio de sesión: de SQL Server, de Windows, asignado a un certificado y asignado a una clave asimétrica. Cuando crea inicios de sesión que se asignan desde una cuenta de dominio de Windows, debe usar el nombre de inicio de sesión de usuario anterior a Windows 2000 con el formato [<Dominio>\<InicioDeSesión>].
PASSWORD **='***password***'**	Sólo se aplica a inicios de sesión de SQL Server. Especifica la contraseña del inicio de sesión que se está creando. Debe usar siempre una contraseña segura.
DEFAULT_DATABASE **=***database*	Especifica la base de datos predeterminada que debe asignarse al inicio de sesión. Si no se incluye esta opción, la base de datos predeterminada es master.
DEFAULT_LANGUAGE **=***language*	Especifica el idioma predeterminado que debe asignarse al inicio de sesión. Si no se incluye esta opción, el idioma predeterminado es el del servidor. Si el idioma predeterminado del servidor se cambia más tarde, el del inicio de sesión se mantiene igual.
CHECK_EXPIRATION **=** { ON | OFF }	Sólo se aplica a inicios de sesión de SQL Server. Especifica si debe aplicarse la directiva de expiración de contraseñas en este inicio de sesión. El valor predeterminado es OFF.
CHECK_POLICY **=** { ON | OFF }	Sólo se aplica a inicios de sesión de SQL Server. Especifica que se deben aplicar las directivas de contraseñas de Windows en el equipo que ejecuta SQL Server para este inicio de sesión. El valor predeterminado es ON.

En el ejemplo siguiente, crea un inicio de sesión para un usuario determinado y le asigna un password. La opción MUST_CHANGE exige a los usuarios que cambien el password la primera vez que se conecten al servidor.

```
CREATE LOGIN cibertec
WITH PASSWORD = '12345678'
MUST_CHANGE;
GO
```

En el ejemplo siguiente, crea un inicio de sesión para un usuario determinado y le asigna un password, la opción DEFAULT_DATABASE predeterminada.

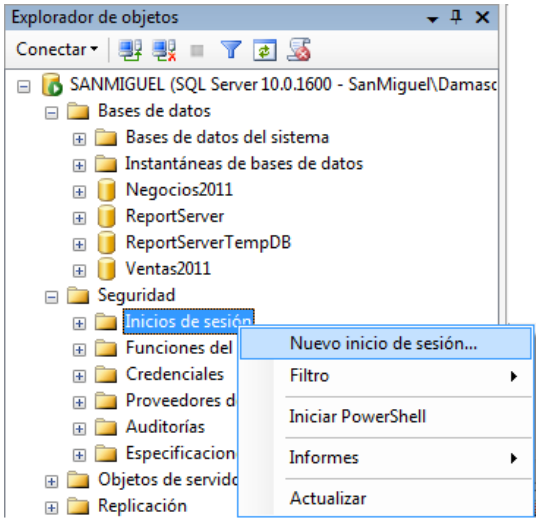
```
CREATE LOGIN cibertec
WITH PASSWORD = 'pa$$w0rd' ,
DEFAULT_DATABASE=Negocios2011
GO
```

Se puede modificar un inicio de sesión o login ejecutando la sentencia ALTER LOGIN. En el ejemplo siguiente, modificamos el password o clave del inicio de sesión cibertec, tal como se muestra.

```
ALTER LOGIN cibertec
WITH PASSWORD = 'upc_cibertec'
GO
```

Podemos remover un login ejecutando la sentencia DROP LOGIN, tal como se muestra en la sentencia.

```
DROP LOGIN cibertec
GO
```

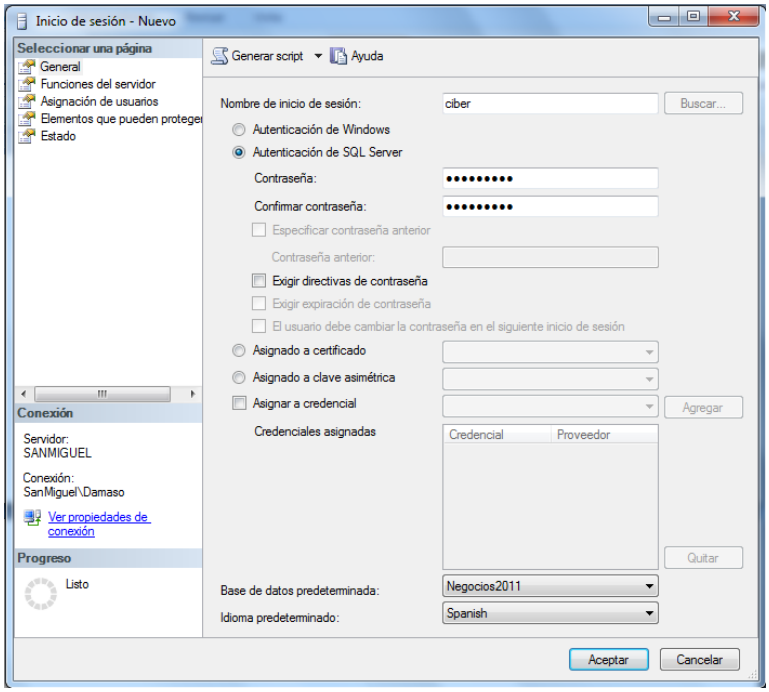


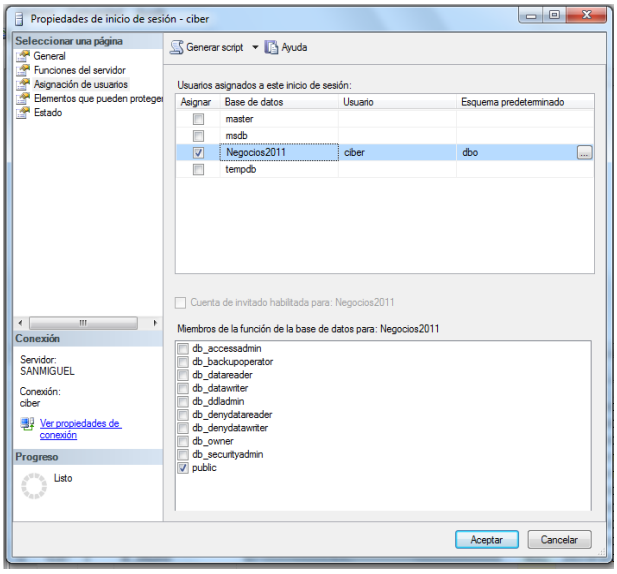
Utilizando el asistente del administrador podemos crear un inicio de sesión.

Desde el explorador de objetos, despliegue la opción SEGURIDAD, en la opción Inicios de sesión, hacer click derecho y seleccione la opción **Nuevo inicio de sesión…**

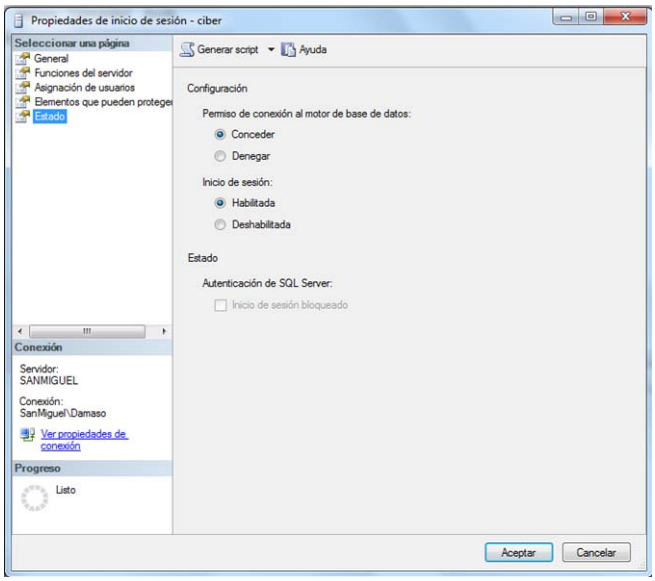
En la ventana inicio de sesión:

- Asigne un nombre al inicio de sesión
- Defina la autenticación, se recomienda que sea de tipo SQL Server
- Escriba y confirme el password
- Desmarcar la opción **Exigir la expiración de la contraseña**
- Seleccione la base de datos predeterminada (Negocios2011), si no lo selecciona, por defecto es la base de datos master.
- Seleccione el idioma.





En la opción **Asignación de usuarios**, asigne al inicio de sesión la base de datos así como el esquema donde trabajará el inicio de sesión, tal como se muestra en la figura.



En la opción Estada, habilitar la sesión para trabajar con el inicio de sesión al momento de realizar la conexión. A continuación, presione el botón ACEPTAR.

5.1.3.2 Creación de usuarios en SQL Server

Se puede crear un usuario utilizando el Administrador corporativo o comandos TRANSACT-SQL.

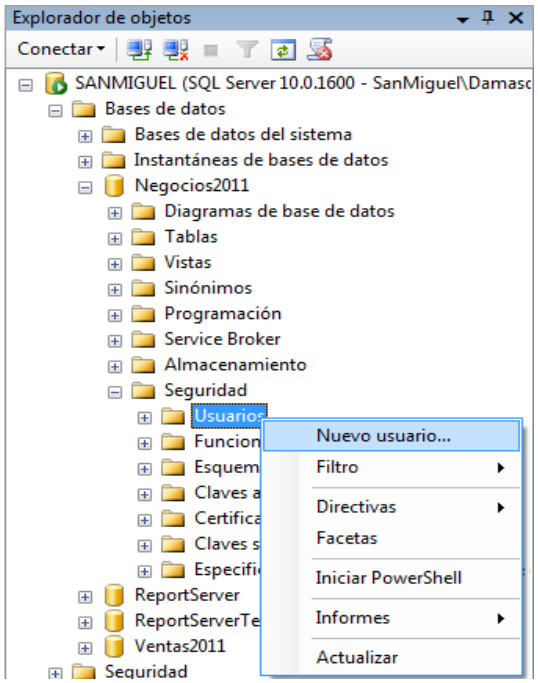
Para crear un usuario utilizando comandos TRANSACT-SQL, utilice el comando CREATE USER, cuya sintaxis es la siguiente:

```
CREATE USER user_name
    [ { { FOR | FROM }
      {
        LOGIN login_name
      }
      | WITHOUT LOGIN
    ]
    [ WITH DEFAULT_SCHEMA = schema_name ]
```

Argumentos	Descripción
user_name	Especifica el nombre por el que se identifica al usuario en esta base de datos. *user_name* es de tipo sysname. Puede tener una longitud máxima de 128 caracteres.
LOGIN *login_name*	Especifica el inicio de sesión de SQL Server del usuario de base de datos que se va a crear. *login_name* debe ser un inicio de sesión válido en el servidor. Cuando este inicio de sesión de SQL Server se introduzca en la base de datos adquirirá el nombre y el identificador del usuario de la base de datos que se va a crear.
WITHOUT LOGIN	Especifica que el usuario no se debe asignar a un inicio de sesión existente

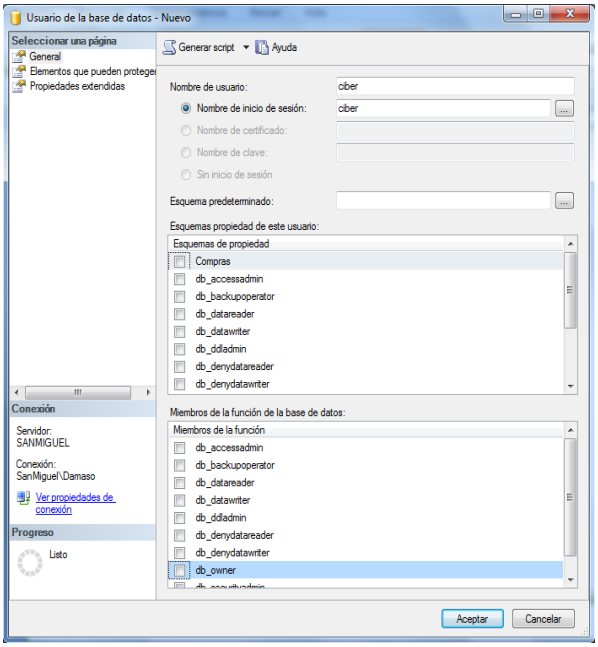
En el siguiente ejemplo, vamos a crear primero un inicio de sesión de servidor denominado BDCibertec con su password, y a continuación, se crea el usuario de base de datos BDCibertec correspondiente a la base de datos Negocios2011

```
CREATE LOGIN BDCIBERTEC
    WITH PASSWORD = 'PA$$W0RD',
    DEFAULT_DATABASE=NEGOCIOS2011;
GO

USE NEGOCIOS2011;
GO

CREATE USER BDCIBERTEC
FOR LOGIN BDCIBERTEC;
GO
```



Utilizando el explorador de objetos, podemos crear un usuario

- Seleccione la base de datos para crear el usuario: Negocios2011
- Desplegar la base de datos, seleccione la carpeta Seguridad.
- Despliegue Seguridad, hacer click derecho en la opción Usuarios, seleccione la opción **Nuevo Usuarios…** tal como se muestra en la figura.



En la ventana Nuevo usuario, asigne el nombre del usuario, asigne el inicio de sesión del usuario. Presione el botón ACEPTAR, para crear el usuario de la base de datos Negocios2011.

Para verificar si un usuario se encuentra en la lista de usuarios de la base de datos, ejecutar una consulta a la tabla **sys.sysusers**, tal como se muestra.

```
USE NEGOCIOS2011;
GO
```

```
SELECT *
FROM SYS.SYSUSERS
GO
```

5.1.3.3 Manejo de Roles a nivel servidor

Para administrar con facilidad los permisos en el servidor, SQL Server proporciona varios *roles*, que son las entidades de seguridad que agrupan a otras entidades de seguridad. Los *roles* son como los *grupos* del sistema operativo Microsoft Windows.

Los roles de nivel de servidor también se denominan *roles fijos de servidor* porque no se pueden crear nuevos roles de nivel de servidor. Puede agregar inicios de sesión de SQL Server, cuentas de Windows y grupos de Windows a los roles de nivel de servidor. Cada miembro de un rol fijo de servidor puede agregar otros inicios de sesión a ese mismo rol.

En la tabla siguiente se muestran los roles de nivel de servidor y sus capacidades

rol de servidor	Descripción
sysadmin	Los miembros del rol fijo de servidor sysadmin pueden realizar cualquier actividad en el servidor.
serveradmin	Los miembros del rol fijo de servidor serveradmin pueden cambiar las opciones de configuración en el servidor y apagarlo.
securityadmin	Los miembros del rol fijo de servidor securityadmin administran los inicios de sesión y sus propiedades. Pueden administrar los permisos de servidor GRANT, DENY y REVOKE. También, pueden administrar los permisos de nivel de base de datos GRANT, DENY y REVOKE si tienen acceso a una base de datos. Asimismo, pueden restablecer las contraseñas para los inicios de sesión de SQL Server. **Nota de seguridad.-** La capacidad de conceder acceso a Motor de base de datos y configurar los permisos de usuario permite que el administrador de seguridad asigne la mayoría de los permisos de

	servidor. El rol **securityadmin** se debe tratar como equivalente del rol **sysadmin**.
processadmin	Los miembros del rol fijo de servidor processadmin pueden finalizar los procesos que se ejecuten en una instancia de SQL Server.
setupadmin	Los miembros del rol fijo de servidor setupadmin pueden agregar y quitar servidores vinculados.
bulkadmin	Los miembros del rol fijo de servidor bulkadmin pueden ejecutar la instrucción BULK INSERT.
diskadmin	El rol fijo de servidor diskadmin se usa para administrar archivos de disco.
dbcreator	Los miembros del rol fijo de servidor dbcreator pueden crear, modificar, quitar y restaurar cualquier base de datos.
public	Cada inicio de sesión de SQL Server pertenece al rol public de servidor. Cuando a una entidad de seguridad de servidor no se le han concedido ni denegado permisos específicos para un objeto protegible, el usuario hereda los permisos concedidos al rol public en ese objeto. Solo asigne los permisos públicos en objetos cuando desee que estos estén disponibles para todos los usuarios.

5.1.3.4 Manejo de Roles a nivel base de datos

Los roles de nivel de base de datos se aplican a toda la base de datos en lo que respecta a su ámbito de permisos.

Existen dos tipos de roles de nivel de base de datos en SQL Server: los *roles fijos de base de datos*, que están predefinidos en la base de datos, y los *roles flexibles de base de datos*, que pueden crearse.

Los roles fijos de base de datos se definen en el nivel de base de datos y existen en cada una de ellas. Los miembros de los roles de base de datos **db_owner** y

db_securityadmin pueden administrar los miembros de los roles fijos de base de datos. Sin embargo, sólo los miembros del rol de base de datos **db_owner** pueden agregar miembros al rol fijo de base de datos **db_owner**.

Puede agregar cualquier cuenta de la base de datos y otros roles de SQL Server a los roles de nivel de base de datos. Cada miembro de un rol fijo de base de datos puede agregar otros inicios de sesión a ese mismo rol.
En la tabla siguiente, se muestran los roles fijos de nivel de base de datos y sus capacidades. Estos roles existen en todas las bases de datos.

rol de base de datos	Descripción
db_owner	Los miembros del rol fijo de base de datos **db_owner** pueden realizar todas las actividades de configuración y mantenimiento en la base de datos y también pueden quitar la base de datos.
db_securityadmin	Los miembros del rol fijo de base de datos **db_securityadmin** pueden modificar la pertenencia a roles y administrar permisos. Si se agregan entidades de seguridad a este rol, podría habilitarse un aumento de privilegios no deseado.
db_accessadmin	Los miembros del rol fijo de base de datos **db_accessadmin** pueden agregar o quitar el acceso a la base de datos para inicios de sesión de Windows, grupos de Windows e inicios de sesión de SQL Server.
db_backupoperator	Los miembros del rol fijo de base de datos **db_backupoperator** pueden crear copias de seguridad de la base de datos.
db_ddladmin	Los miembros del rol fijo de base de datos **db_ddladmin** pueden ejecutar cualquier comando del lenguaje de definición de datos (DDL) en una base de datos.
db_datawriter	Los miembros del rol fijo de base de datos **db_datawriter** pueden agregar, eliminar o cambiar datos en todas las tablas de usuario.
db_datareader	Los miembros del rol fijo de base de datos **db_datareader**

	pueden leer todos los datos de todas las tablas de usuario.
db_denydatawriter	Los miembros del rol fijo de base de datos **db_denydatawriter** no pueden agregar, modificar ni eliminar datos de tablas de usuario de una base de datos.
db_denydatareader	Los miembros del rol fijo de base de datos **db_denydatareader** no pueden leer datos de las tablas de usuario dentro de una base de datos.

5.1.4 Permisos en el motor de base de datos

Todos los elementos protegibles de SQL Server tienen permisos asociados que se pueden conceder a una entidad de seguridad.

Los permisos se pueden manipular con las conocidas consultas GRANT, DENY y REVOKE de TRANSACT-SQL.

5.1.4.1 Comando GRANT

Concede permisos sobre un elemento protegible a una entidad de seguridad. El concepto general es GRANT <algún permiso> ON <algún objeto> TO <algún usuario, inicio de sesión o grupo>.
La sintaxis de este comando:

```
GRANT { ALL [ PRIVILEGES ] }
      | permission [ ( column [ ,...n ] ) ] [ ,...n ]
      [ ON [ class :: ] securable ] TO principal [ ,...n ]
      [ WITH GRANT OPTION ] [ AS principal ]
```

La sintaxis completa de la instrucción GRANT es compleja. El anterior diagrama de sintaxis se ha simplificado para concentrar la atención en su estructura. La sintaxis completa para conceder permisos para elementos protegibles específicos se describe en los temas enumerados a continuación.

En la siguiente tabla, se enumeran los elementos protegibles y los temas donde se describe la sintaxis específica de los mismos.

Rol de aplicación	GRANT (permisos de entidad de seguridad de base de datos de Transact-SQL)
Certificado	GRANT (permisos de certificado de Transact-SQL)
Base de datos	GRANT (permisos de base de datos de Transact-SQL)
Función	GRANT (permisos de objeto de Transact-SQL)
Inicio de sesión	GRANT (permisos de entidad de seguridad de servidor de Transact-SQL)
Objeto	GRANT (permisos de objeto de Transact-SQL)
Rol	GRANT (permisos de entidad de seguridad de base de datos de Transact-SQL)
Esquema	GRANT (permisos de esquema de Transact-SQL)
Servidor	GRANT (permisos de servidor de Transact-SQL)
Procedimiento almacenado	GRANT (permisos de objeto de Transact-SQL)
Objetos de sistema	GRANT (permisos de objeto de sistema de Transact-SQL)
Tabla	GRANT (permisos de objeto de Transact-SQL)
Tipo	GRANT (permisos de tipo de Transact-SQL)
Usuario	GRANT (permisos de entidad de seguridad de base de datos de Transact-SQL)
Vista	GRANT (permisos de objeto de Transact-SQL)

El siguiente ejemplo muestra cómo otorgar permiso de alterar cualquier base de datos al usuario ciber

```
GRANT ALTER ANY DATABASE
TO ciber
```

El siguiente ejemplo muestra como otorgar permiso de ejecutar el comando SELECT y UPDATE hacia la tabla Ventas.Cliente, al usuario ciber

```
GRANT SELECT,UPDATE
ON Ventas.Clientes
TO ciber
```

5.1.4.2 Comando DENY

Deniega un permiso a una entidad de seguridad. Evita que la entidad de seguridad herede permisos por su pertenencia a grupos o roles. La sintaxis simplificada de este comando:

```
 DENY { ALL [ PRIVILEGES ] }
      | permission [ ( column [ ,...n ] ) ] [ ,...n ]
      [ ON [ class :: ] securable ] TO principal [ ,...n ]
      [ CASCADE] [ AS principal ]
```

La sintaxis completa de la instrucción DENY es compleja. El diagrama anterior se ha simplificado para concentrar la atención en su estructura.

DENY producirá un error si CASCADE no se especifica al denegar un permiso a una entidad de seguridad a la que se concedió ese permiso con GRANT OPTION.

El siguiente ejemplo muestra cómo denegar permiso de crear un procedimiento almacenado al usuario ciber

```
DENY CREATE PROCEDURE
TO ciber
```

El siguiente ejemplo muestra como denegar permiso de ejecutar el comando UPDATE y DELETE hacia la tabla RRHH.Empleados, al usuario ciber

```
DENY UPDATE, DELETE
ON RRHH.Empleados
TO ciber
```

5.1.4.3 Comando REVOKE

Quita un permiso concedido o denegado previamente. La sintaxis simplificada de este comando:

```
 REVOKE [ GRANT OPTION FOR ]   {
        [ ALL [ PRIVILEGES ] ]
        | permission [ ( column [ ,...n ] ) ] [ ,...n ]
      }
      [ ON [ class :: ] securable ]
      { TO | FROM } principal [ ,...n ]
      [ CASCADE] [ AS principal ]
```

La sintaxis completa de la instrucción REVOKE es compleja. El diagrama de sintaxis anterior se ha simplificado para concentrar la atención en su estructura.

El siguiente ejemplo muestra como revocar permiso de crear una tabla al usuario ciber

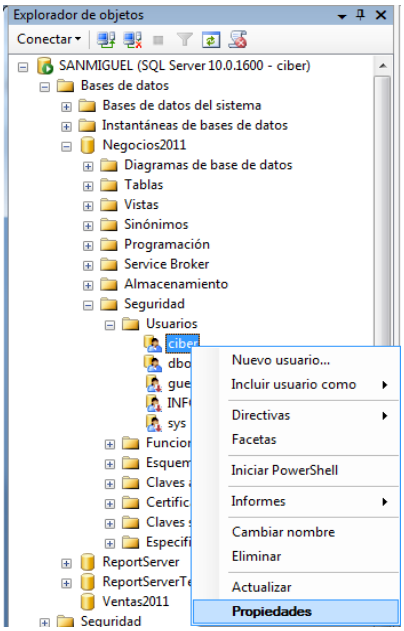
```
REVOKE CREATE TABLE
FROM ciber
```

El siguiente ejemplo muestra como revocar permiso de ejecutar el comando SELECT, UPDATE y DELETE hacia la tabla RRHH.Empleados, al usuario ciber

```
REVOKE SELECT, UPDATE, DELETE
ON RRHH.Empleados
TO ciber
go
```

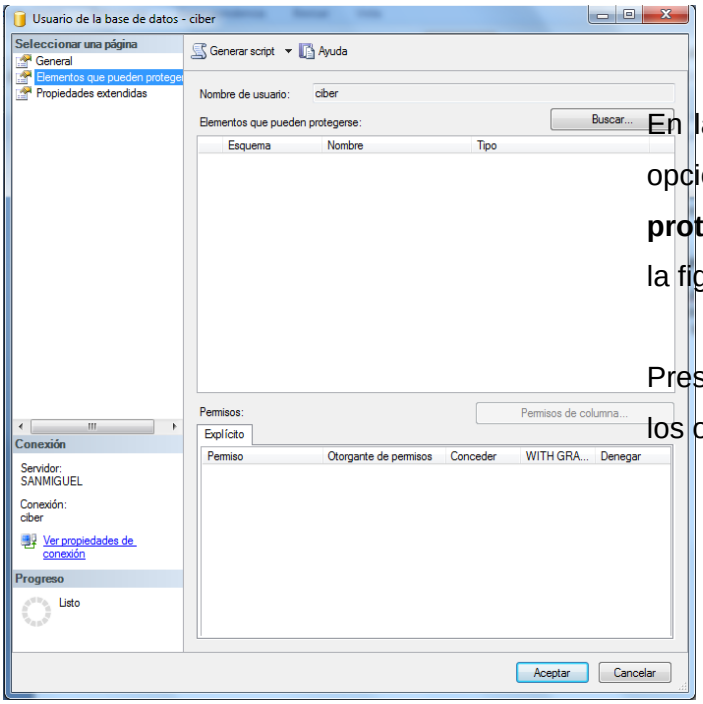
5.1.4.4 Asignando Permisos al usuario utilizando el Administrador Corporativo

El administrador de una base de datos puede otorgar, denegar o revocar permisos al usuario sobre los objetos de la base de datos asignada.



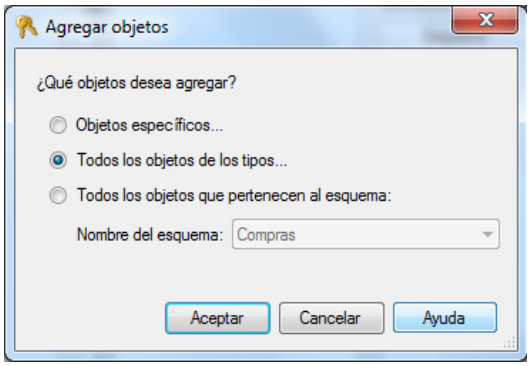
Para realizar este proceso, seleccione la base de datos, para nuestro case será Negocios2011, tal como se muestra.

Seleccione la carpeta SEGURIDAD, desplegar la carpeta.
Seleccione la carpeta USUARIOS, hacer click derecho sobre la carpeta y seleccione la opción PROPIEDADES, tal como se muestra en la figura.



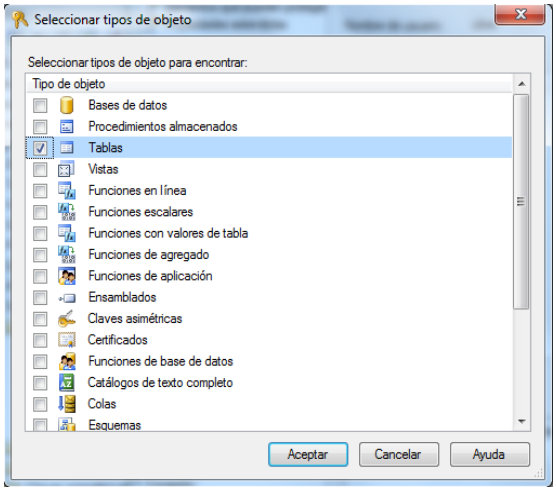
En la ventana Usuario, seleccione la opción **Elementos que pueden protegerse** tal como se muestra en la figura.

Presione el botón **Buscar** para elegir los objetos a proteger.



En esta opción, debemos seleccionar los objetos que vamos a asignar permisos.

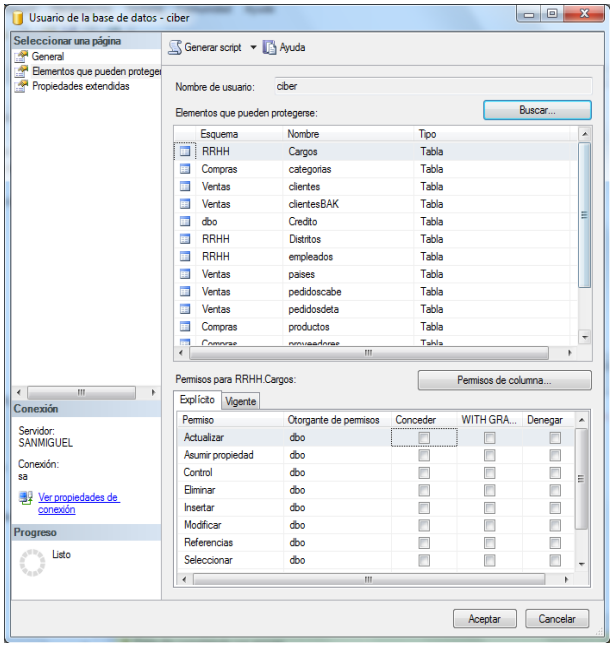
Seleccione todos los objetos para que liste los objetos de la base de datos. Presione el botón ACEPTAR.



Seleccione los tipos de objetos que vamos a manejar sus permisos.

Para nuestro caso, seleccionamos el objeto TABLAS, tal como se muestra.

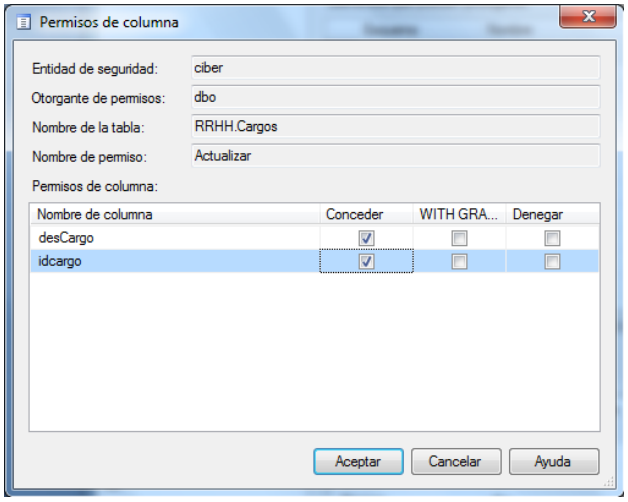
Presione el botón ACEPTAR.



A continuación, se lista todas las tablas de la base de datos.

Al seleccionar una tabla, se listan los permisos de la tabla para cada uno de sus procesos.

Incluso, podemos controlar permisos a cada columna de la tabla.

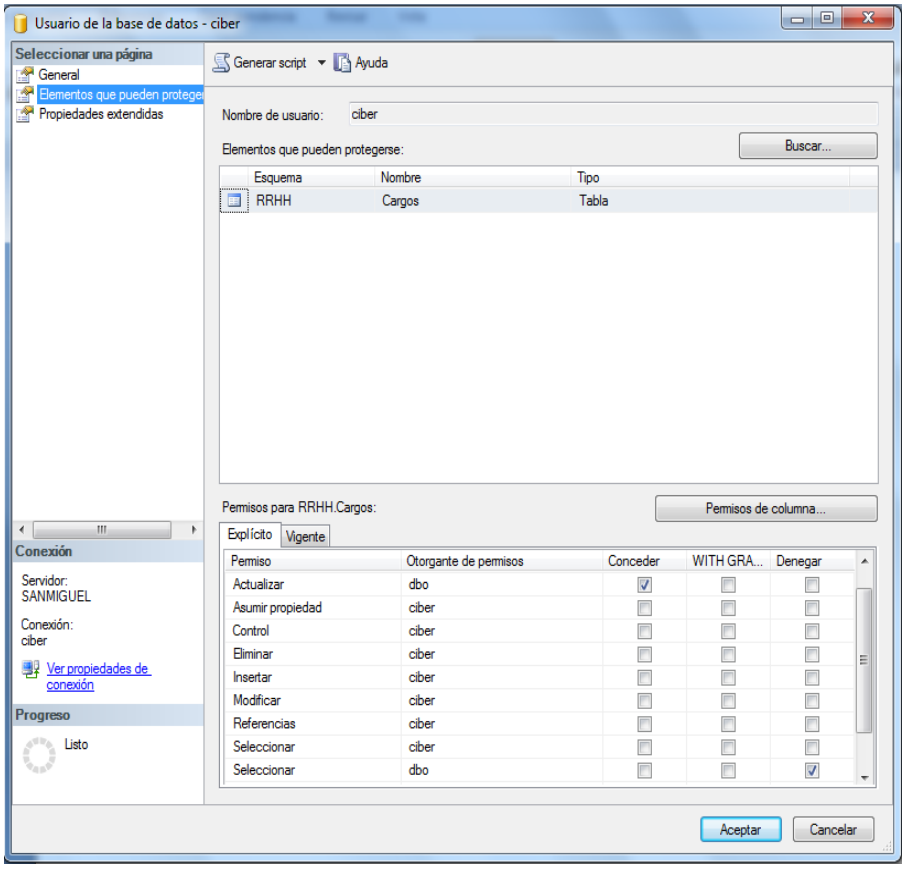


En esta ventana, podemos dar permisos a las columnas de la tabla en un determinado proceso (ACTUALIZAR)

Para confirmar el proceso, presione el botón ACEPTAR.

Para confirmar los permisos, presione el botón ACEPTAR de la ventana principal.

Si abrimos la ventana de propiedades del usuario, visualizamos los permisos efectuado a la tabla RRHH.Cargos que concede el permiso de UPDATE y denega el permiso de SELECT



Resumen

- Las *entidades de seguridad* son entidades que pueden solicitar recursos de SQL Server. Igual que otros componentes del modelo de autorización de SQL Server, las entidades de seguridad se pueden organizar en jerarquías.
- El inicio de sesión **sa** de SQL Server es una entidad de seguridad del servidor. Se crea de forma predeterminada cuando se instala una instancia. En SQL Server 2005 y SQL Server 2008, la base de datos predeterminada de **sa** es **master**. Es un cambio de comportamiento con respecto a versiones anteriores de SQL Server.
- Por definición, un cliente y un servidor de base de datos son entidades de seguridad y se pueden proteger. Estas entidades se pueden autenticar mutuamente antes de establecer una conexión de red segura. SQL Server admite el protocolo de autenticación Kerberos, que define cómo interactúan los clientes con un servicio de autenticación de red.
- Para acceder a SQL SERVER, se dispone de dos modos de autenticación: Autenticación de Windows y Autenticación en modo mixto. En la autenticación de Windows, el sistema operativo es el responsable de identificar al usuario. SQL Server usa después esa identificación del sistema operativo para determinar los permisos del usuario que hay que aplicar. Con la autenticación en modo mixto, los dos Windows y SQL Server son responsables de identificar al usuario.
- Una cuenta de usuario de Windows puede ser necesaria para conectarse a una base de datos. También, podría ser necesaria la autenticación de SQL Server. Tanto si se va a usar autenticación de Windows o autenticación de modo mixto, la cuenta que se usa para la conexión a SQL SERVER se le conoce como inicio de sesión de SQL Server.
- Una consulta SELECT devuelve los resultados como un conjunto de filas. Opcionalmente, se pueden recuperar resultados formales de una consulta SQL como XML especificando la cláusula FOR XML en la consulta. La cláusula FOR XML puede usarse en consultas de nivel superior y en subconsultas.
- Para administrar con facilidad los permisos en el servidor, SQL Server proporciona varios *roles*, que son las entidades de seguridad que agrupan a otras entidades de seguridad. Los *roles* son como los *grupos* del sistema operativo Microsoft Windows

- Los roles de nivel de base de datos se aplican a toda la base de datos en lo que respecta a su ámbito de permisos. Existen dos tipos de roles de nivel de base de datos en SQL Server: los *roles fijos de base de datos*, que están predefinidos en la base de datos, y los *roles flexibles de base de datos*, que pueden crearse.
- Todos los elementos protegibles de SQL Server tienen permisos asociados que se pueden conceder a una entidad de seguridad. Los permisos se pueden manipular con las conocidas consultas GRANT, DENY y REVOKE de TRANSACT-SQL
- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas:

  - http://msdn.microsoft.com/es-es/library/ms173463.aspx

    Aquí hallará los conceptos de crear usuarios
  - http://mredison.wordpress.com/2009/03/15/como-creo-un-usuario-en-sql-server/

    En esta página, hallará los conceptos de cómo crear un usuario
  - http://msdn.microsoft.com/es-es/library/bb510418.aspx

    Aquí hallará los conceptos de manejo identidades y control de acceso

UNIDAD DE APRENDIZAJE
6

SEGURIDAD Y RESTAURACIÓN EN SQL SERVER

LOGRO DE LA UNIDAD DE APRENDIZAJE

- Al término de la unidad, el alumno haciendo uso de los conocimientos de mejores prácticas en la copia de seguridad y restauración de una base de datos, realizará copias de respaldo o restauración de una base de datos en SQL SERVER.

TEMARIO

1.1. Introducción a las estrategias de seguridad y restauración en SQL Server
1.1.1. Impacto del modelo de recuperación de copia de seguridad.
1.1.2. Diseño de la estrategia de copia de seguridad
1.2. Copia de Seguridad en SQL Server
1.2.1. Copia de seguridad completa
1.2.2. Copia de seguridad diferencial
1.2.3. Copia de seguridad de registro de transacciones
1.2.4. BACK UP (Transact-SQL)
1.2.5. Realizar copia de seguridad utilizando el Management Studio de SQL Server
1.3. Restaurando una copia de seguridad
1.3.1. RESTORE (Transact-SQL)
1.3.2. Restaurando una base de datos utilizando Management Studio

ACTIVIDADES PROPUESTAS

- Los alumnos crea copia de respaldo o seguridad a una base de datos y su registro de transacciones en forma completa o diferencial.
- Los alumnos restauran una base de datos desde una copia de respaldo o de seguridad de una base de datos o de registro de transacciones.

6.1 INTRODUCCION A LAS ESTRATEGIAS DE SEGURIDAD Y RESTAURACION EN SQL SERVER

El propósito de crear copias de seguridad de SQL Server es para que usted pueda recuperar una base de datos dañada. Sin embargo, copias de seguridad y restauración de los datos debe ser personalizado para un ambiente particular y debe trabajar con los recursos disponibles. Por lo tanto, un uso fiable de copia de seguridad y restauración para la recuperación exige una copia de seguridad y restauración de la estrategia.

Una copia de seguridad bien diseñado y restaurar la estrategia maximiza la disponibilidad de datos y minimiza la pérdida de datos, teniendo en cuenta sus necesidades de negocio en particular.

Una estrategia de copia de seguridad y restauración de copia de seguridad contiene una porción de seguridad y una porción de restauración. La parte de seguridad de la estrategia define el tipo y frecuencia de las copias de seguridad, la naturaleza y la velocidad del hardware que se requiere para que, como copias de seguridad deben ser probados, y dónde y cómo los medios de comunicación copia de seguridad se van a almacenar (incluyendo las consideraciones de seguridad).

La parte de restauración de la estrategia define quién es responsable de las restauraciones y cómo restaurar se debe realizar para cumplir con sus objetivos de disponibilidad de la base de datos y para minimizar la pérdida de datos. Le recomendamos que documente la copia de seguridad y restaurar los procedimientos y guardar una copia de la documentación en su libro de ejecutar.

6.1.1 Impacto del modelo de recuperación de copia de seguridad y restauración

Copia de seguridad y restauración de las operaciones se producen en el contexto de un modelo de recuperación. Un modelo de recuperación es una propiedad de base de datos que controla el registro de transacciones que se gestiona. Además, el modelo de recuperación de una base de datos determina qué tipos de copias de seguridad y

restauración son compatibles con la base de datos. Normalmente, una base de datos utiliza ya sea el modelo de recuperación simple o el modelo de recuperación completa.

La mejor opción de modelo de recuperación de la base de datos depende de los requerimientos de su negocio. Para evitar la gestión del registro de transacciones y simplificar el BACKUP y restauración, utilice el modelo de recuperación simple. Para minimizar la pérdida de trabajo, a costa de los gastos generales de administración, utilice el modelo de recuperación completa.

6.1.2 Diseño de la estrategia de copia de seguridad

Seleccionado un modelo de recuperación que se ajuste a sus requisitos de negocio para una base de datos específica, usted tiene que planificar e implementar una estrategia de copia de seguridad correspondiente.

La estrategia de copia de seguridad óptima depende de una variedad de factores, de los cuales los siguientes son especialmente importantes:

- ¿Cuántas horas al día no tiene aplicaciones para acceder a la base de datos?

Si hay una predecible temporada baja período, le recomendamos que programe copias de seguridad completas de bases de datos para ese período.

- ¿Con qué frecuencia se producen cambios y actualizaciones probables que ocurra?

Si los cambios son frecuentes, considere lo siguiente:

- Bajo el modelo de recuperación simple, considere la programación de copias de seguridad diferenciales entre copias de seguridad de base de datos completa.
- Bajo el modelo de recuperación completa, debe programar copias de seguridad frecuentes de registro. Programación de copias de seguridad diferenciales entre copias de seguridad completas puede reducir el tiempo de restauración mediante la reducción del número de copias de seguridad de registros que se deben restaurar después de restaurar los datos.
- Son los cambios probables de ocurrir en tan sólo una pequeña parte de la base de datos o en una gran parte de la base de datos?

Para una gran base de datos en la que los cambios se concentran en una parte de los archivos o grupos de archivos, copias de seguridad parciales o copias de seguridad de archivos y puede ser útil.

- ¿Cuánto espacio en disco una copia de seguridad completa requiere?

6.1.3 Copia de seguridad en SQL SERVER

El alcance de una copia de seguridad de los datos puede ser una base de datos completa, una base de datos parciales, o un conjunto de archivos o grupos de archivos. Para cada uno de estos, SQL Server admite copias de seguridad completas y diferenciales:

6.1.3.1 Copia de seguridad completa

Contiene todos los datos en una base de datos específica o un conjunto de grupos de archivos o archivos, y también suficiente registro para permitir la recuperación de los datos.

6.1.3.2 Copia de seguridad diferencial

Se basa en la última copia de seguridad completa de los datos. Esto se conoce como copia diferencial. Una copia diferencial es una copia de seguridad completa de lectura / escritura de datos. Una copia de seguridad diferencial incluye sólo los datos que ha cambiado desde la base diferencial. Por lo general, los respaldos diferenciales que se toman poco después de la copia de seguridad de base son más pequeños y más rápidos de crear que la base de una copia de seguridad completa. Por lo tanto, utilizar los respaldos diferenciales puede acelerar el proceso de hacer copias de seguridad frecuentes para disminuir el riesgo de pérdida de datos. Por lo general, una base diferencial es utilizada por varias copias de seguridad diferenciales sucesivas. En el tiempo de restauración, la copia de seguridad que se restaura el primero, seguido por la copia de seguridad diferencial más reciente.

6.1.3.3 Copia de seguridad del registro de transacciones

Bajo el modelo de recuperación optimizado para cargas masivas de registros de modelo de recuperación, *copias de seguridad del registro de transacciones* (o *copias de seguridad de registro)* son obligatorias. Cada copia de seguridad de registro cubre la parte del registro de transacciones que estaba activa cuando la copia de seguridad fue creada, e incluye todos los registros que no fueron respaldados en una copia de seguridad de registros anterior. Una secuencia ininterrumpida de seguridad del registro contiene la cadena de registro completo de la base de datos, lo que se dice que es continua. Bajo el modelo de recuperación completa, y, a veces bajo el registro masivo modelo de recuperación, una cadena de registros ininterrumpida permite restaurar la base de datos a cualquier punto en el tiempo.

Antes de crear la copia de seguridad de registro en primer lugar, debe crear una copia de seguridad completa, como una copia de seguridad de bases de datos. A partir de entonces, la copia de seguridad del registro de transacciones con regularidad es necesaria, no sólo para minimizar la pérdida de trabajo sino también para permitir el truncamiento del registro de transacciones.

.

6.1.3.4 BACKUP (TRANSACT-SQL)

Comando TRANSACT-SQL que realiza copias de seguridad de una base de datos completa, o uno o más archivos o grupos de archivos (BASE DE DATOS DE SEGURIDAD). Además, bajo el modelo de recuperación optimizado para cargas masivas de registros de modelo de recuperación, copias de seguridad del registro de transacciones (BACKUP LOG).

La sintaxis para realizar un BACK UP a la base de datos es la siguiente:

```
BACKUP DATABASE { nombre_base_datos | @var_nombre_base_datos }
  TO <dispositivo_backup> [ ,...n ]
  [ <MIRROR TO clause> ] [ next-mirror-to ]
  [ WITH { DIFFERENTIAL | <general_WITH_options> [ ,...n ] } ]
[;]
```

La sintaxis para realizar un BACK UP a los Filegroup de la base de datos

```
BACKUP DATABASE { nombre_base_datos | @var_nombre_base_datos }
  <file_or_filegroup> [ ,...n ]
  TO <dispositivo_backup> [ ,...n ]
  [ <MIRROR TO clause> ] [ next-mirror-to ]
  [ WITH { DIFFERENTIAL | <general_WITH_options> [ ,...n ] } ]
[;]
```

La sintaxis para realizar un BACK UP al registro de transacciones o LOG de la base de datos

```
BACKUP LOG { nombre_base_datos | @var_nombre_base_datos }
  TO <dispositivo_backup> [ ,...n ]
  [ <MIRROR TO clause> ] [ next-mirror-to ]
  [ WITH { <general_WITH_options> | <log-specific_optionspec> } [
,...n ] ]
 [;]
```

Argumentos

Argumento	Descripción
DATABASE	Especifica una copia de seguridad completa de la base de datos. Si se especifica una lista de archivos y grupos de archivos, sólo se realiza la copia de seguridad de esos archivos o grupos de archivos. Durante una copia de seguridad completa o diferencial de una base de datos, SQL Server realiza la copia de seguridad de una parte suficiente del registro de transacciones para producir una base de datos coherente cuando se restaure la base de datos.
LOG	Especifica que sólo se realizará la copia de seguridad del registro de transacciones. Se realiza la copia de

	seguridad del registro desde la última copia de seguridad del registro ejecutada correctamente hasta el final actual del registro.
{ nombre_base_datos | @var_nombre_base_datos }	Es la base de datos para la que se realiza la copia de seguridad del registro de transacciones, de una parte de la base de datos o de la base de datos completa. Si se proporciona como una variable (@var_nombre_base_datos), este nombre se puede especificar como una constante de cadena (@var_nombre_base_datos **=***database name*) o como una variable de un tipo de datos de cadena de caracteres
<file_or_filegroup> [,...*n*]	Se utiliza sólo con BACKUP DATABASE, especifica un grupo de archivos o un archivo de copia de seguridad que se va a incluir en una copia de seguridad de archivos, o especifica un grupo de archivos o un archivo de sólo lectura que se va a incluir en una copia de seguridad parcial. En el modelo de recuperación simple, se permite la copia de seguridad de un grupo de archivos sólo si se trata de un grupo de archivos de sólo lectura.
TO <dispositivo_backup> [,...n]	Indica que el conjunto de dispositivos de copia de seguridad correspondiente es un conjunto de medios no reflejado o el primero de los reflejos de un conjunto de medios reflejado. < dispositivo_backup > Especifica el dispositivo de copia de seguridad físico o lógico que se va a utilizar para la operación de copia de

	seguridad. { device_logico | @var_device_logico } Es el nombre lógico del dispositivo de copia de seguridad en que se hace la copia de seguridad de la base de datos. El nombre lógico debe seguir las reglas definidas para los identificadores. { DISK | TAPE } = { 'device_fisico' | @var_device_fisico } Especifica un archivo de disco o un dispositivo de cinta.
[<MIRROR TO clause>] [next-mirror-to]	Especifica un conjunto de hasta tres dispositivos de copia de seguridad, cada uno de los cuales reflejará los dispositivos de copia de seguridad especificados en la cláusula TO. La cláusula MIRROR TO debe incluir el mismo número y tipo de dispositivos de copia de seguridad que la cláusula TO.

Opciones de WITH

Opción	Descripción
DIFFERENTIAL	Se utiliza sólo con BACKUP DATABASE. Especifica que la copia de seguridad de la base de datos o el archivo sólo debe estar compuesta por las partes de la base de datos o el archivo que hayan cambiado desde la última copia de seguridad completa. Una copia de seguridad diferencial suele ocupar menos espacio que una copia de seguridad completa.
{ NOINIT | INIT }	NOINIT Indica que el conjunto de copia de seguridad se anexa al conjunto de medios especificado, conservando así los conjuntos de copia de seguridad existentes, es el valor predeterminado.

	INIT Especifica que se deben sobrescribir todos los conjuntos de copia de seguridad, pero conserva el encabezado de los medios. Si se especifica INIT, se sobrescriben todos los conjuntos de copia de seguridad existentes en el dispositivo, si las condiciones lo permiten.
{ NOSKIP | SKIP }	NOSKIP Indica a la instrucción BACKUP que compruebe la fecha de expiración de todos los conjuntos de copia de seguridad de los medios antes de permitir que se sobrescriban. Éste es el comportamiento predeterminado. SKIP Deshabilita la comprobación de la expiración y el nombre del conjunto de copia de seguridad que suele realizar la instrucción BACKUP para impedir que se sobrescriban los conjuntos de copia de seguridad. Para obtener más información acerca de las interacciones entre {INIT | NOINIT} y {NOSKIP | SKIP},
{ NOFORMAT | FORMAT }	NOFORMAT Especifica que la operación de copia de seguridad conservará los conjuntos de copias de seguridad y el encabezado del medio existentes en los volúmenes del medio usados en esta operación de copia de seguridad. Éste es el comportamiento predeterminado. FORMAT Especifica que se debe crear un conjunto de medios nuevo. FORMAT hace que la operación de copia de seguridad escriba un nuevo encabezado en todos los volúmenes del medio usados en la operación de copia de seguridad.
{NO_CHECKSUM|CHECKSUM}	NO_CHECKSUM Deshabilita de forma explícita la generación de

	sumas de comprobación de copia de seguridad (y la validación de sumas de comprobación de página). Es el comportamiento predeterminado, salvo para una copia de seguridad comprimida. CHECKSUM Habilita las sumas de comprobación de copia de seguridad.
NO_TRUNCATE	Especifica que el registro no se va a truncar y hace que Motor de base de datos intente hacer la copia de seguridad con independencia del estado de la base de datos. Por consiguiente, una copia de seguridad realizada con NO_TRUNCATE puede tener metadatos incompletos.

El siguiente ejemplo muestra cómo se consigue una copia de seguridad de la base de datos Negocios2011 en forma complete en el disco

```
Use Negocios2011
Go

BACKUP DATABASE Negocios2011
TO DISK = 'D:\SQLNegocios2011.Bak'
WITH FORMAT;
```

El siguiente ejemplo muestra cómo se logra una copia de seguridad de la base de datos Negocios2011 en forma diferencial en el disco

```
Use Negocios2011
Go

BACKUP DATABASE Negocios2011
TO DISK = 'D:\SQLNegocios2011.Bak'
WITH DIFFERENTIAL;
go
```

En el ejemplo siguiente, se crea una copia de seguridad de archivos completa de cada archivo en los dos grupos de archivos secundarios.

```
Use Negocios2011
Go

BACKUP DATABASE Negocios2011
   FILEGROUP = 'SalesGroup1',
   FILEGROUP = 'SalesGroup2'
   TO DISK = 'D:\SQLNegocios2011.Bak'
Go
```

En el siguiente ejemplo, crea una copia de seguridad de archivos diferencial de todos los archivos secundarios.

```
Use Negocios2011
Go

BACKUP DATABASE Negocios2011
   FILEGROUP = 'SalesGroup1',
   FILEGROUP = 'SalesGroup2'
   TO DISK = 'D:\SQLNegocios2011.Bak'
WITH DIFFERENTIAL;
Go
```

En el ejemplo siguiente, se realiza la copia de seguridad de la base de datos de ejemplo Negocios2011, que usa de forma predeterminada un modelo de recuperación simple. Para admitir las copias de seguridad del registro, la base de datos Negocios2011 se ha modificado para usar el modelo de recuperación completa.

A continuación, en el ejemplo se usa sp_addumpdevice para crear un dispositivo de copia de seguridad lógico para realizar la copia de seguridad de datos, NegociosData, y se crea un dispositivo de copia de seguridad lógico para copiar el registro, NegociosLog.

A continuación, en el ejemplo se crea una copia de seguridad de base de datos completa en NegociosData y, tras un periodo de actividad de actualización, se copia el registro en NegociosLog.

```
/*Para permitir copias de seguridad de registro, antes de la copia de
seguridad completa, modificar la base de datos utilice full recovery
model*/

USE Negocios2011;
GO
ALTER DATABASE Negocios2011
   SET RECOVERY FULL;
GO

-- - Crear NegociosData y dispositivos de copia de seguridad
NegociosLog lógica.

EXEC sp_addumpdevice 'disk', 'NegociosData',
'D:\NegociosData.bak';
GO

EXEC sp_addumpdevice 'disk', 'NegociosLog',
'D:\NegociosLog.bak';
GO

-- Copia de seguridad de la base de datos completa Negocios2011.
BACKUP DATABASE Negocios2011 TO NegociosData;
GO
-- Copia de seguridad del registro de Negocios2011.
BACKUP LOG Negocios2011
   TO NegociosLog;
GO
```

En el ejemplo siguiente, se da formato a los medios, que crean un nuevo conjunto de medios y se realiza una copia de seguridad completa comprimida de la base de datos Negocios2011.

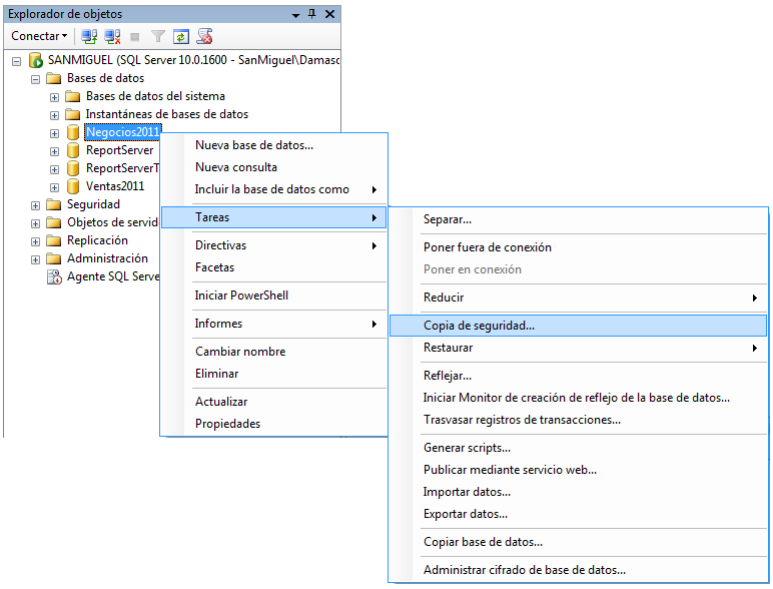
```
Use Negocios2011
Go

BACKUP DATABASE Negocios2011
TO DISK = 'D:\SQLNegocios2011.Bak'
WITH FORMAT, COMPRESSION;
Go
```

En el siguiente ejemplo, se crea un conjunto de medios reflejado que contiene una sola familia de medios y cuatro reflejos, y se realiza una copia de seguridad de la base de datos Negocios2011.

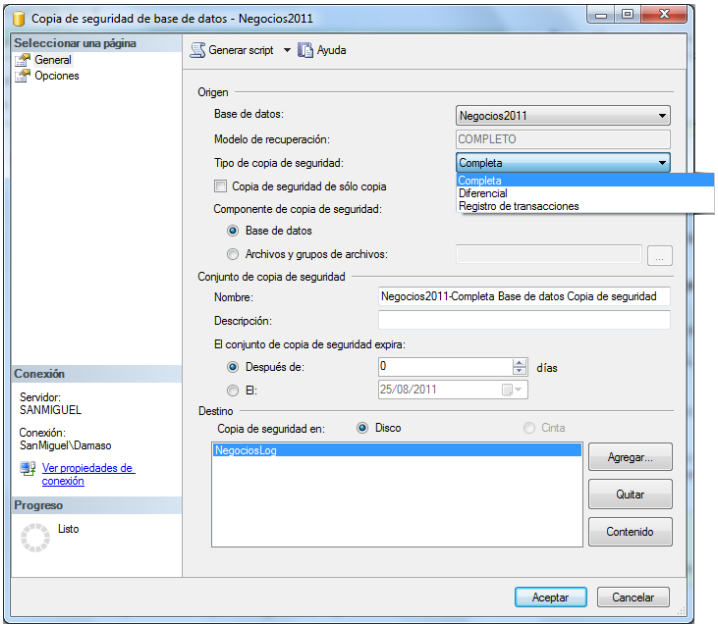
```
BACKUP DATABASE Negocios2011
TO TAPE = '\\.\tape0'
MIRROR TO TAPE = '\\.\tape1'
MIRROR TO TAPE = '\\.\tape2'
MIRROR TO TAPE = '\\.\tape3'
WITH    FORMAT
Go
```

6.1.3.5 Realizar una copia de seguridad utilizando el SQL SERVER Management Studio

1. Después de conectarse a la instancia adecuada del motor de datos de Microsoft SQL Server, en el Explorador de objetos, haga clic en el nombre del servidor para expandir el árbol del servidor.
2. Expanda **Bases de datos,** seleccione una base de datos de usuario.
3. Haga clic derecho en la base de datos, seleccione **Tareas y,** a continuación, haga clic en **copia de seguridad.** La **copia de seguridad de base de datos** aparece el cuadro de diálogo.

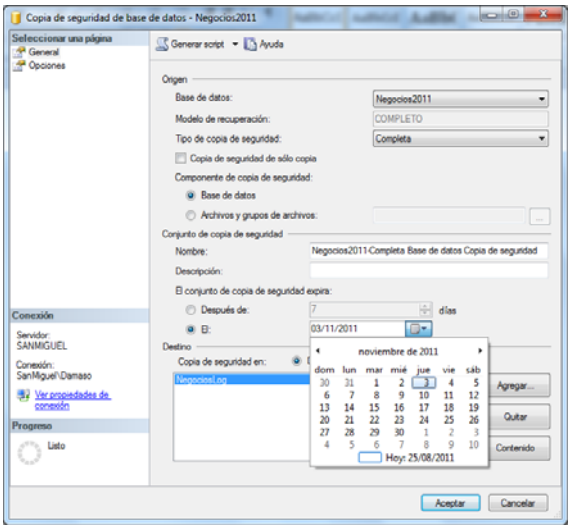


4. En el cuadro de lista **Base de datos,** compruebe el nombre de base de datos. Si lo desea, puede seleccionar otra base de datos de la lista.
5. Usted puede realizar una copia de seguridad de base de datos para cualquier modelo de recuperación **(FULL, BULK_LOGGED,** o **simple).**
6. En el cuadro de lista **Tipo de copia de seguridad,** seleccione **Completa.**

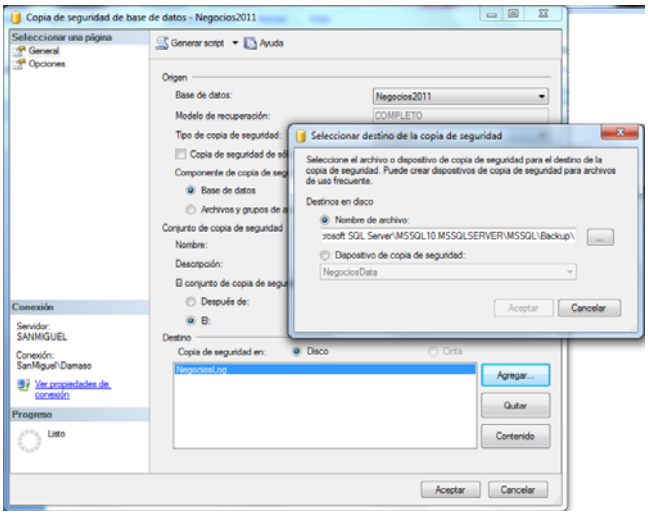


7. Especificar cuando el conjunto de copia de seguridad caduca y pueden ser anuladas sin saltar explícitamente la verificación de los datos de caducidad.

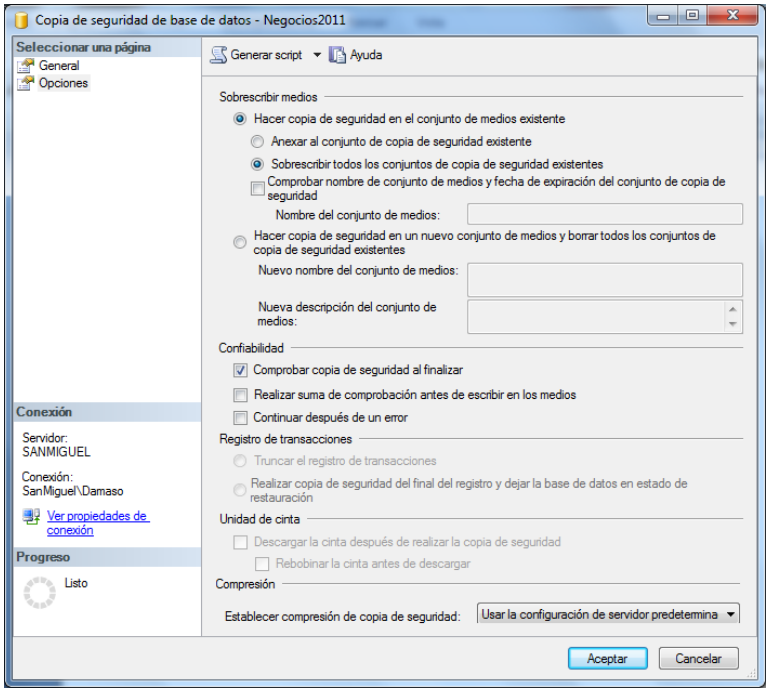
8. Para que el conjunto de copia de seguridad caduque después de un número específico de días, haga clic en **Después de** (opción predeterminada), e introduzca el número de días después de la creación del conjunto en que expirará. Este valor puede ser de 0 a 99999 días; un valor de 0 significa que el conjunto de copia de seguridad nunca se expira.



9. Seleccione el tipo de destino de copia de seguridad, haga clic en **Disco** o **Cinta.** Para seleccionar las rutas de hasta 64 unidades de disco o cinta que contengan un conjunto de medios, haga clic en **Agregar.** Las rutas seleccionadas se muestran en el cuadro de **lista** Copia **de seguridad.** Para eliminar un destino de copia de seguridad, seleccione y haga clic en **Quitar.** Para ver el contenido de un destino de copia de seguridad, seleccione y haga clic en **Contenido.**



10. Seleccionar las opciones avanzadas, haga clic en **Opciones** en el panel **Seleccionar una página.**
11. Seleccione una opción de **Sobrescribir medios de comunicación,** haciendo clic en: **Copia de seguridad en el conjunto de medios existente**
12. En la sección de **fiabilidad,** de manera opcional de verificación:
    - **Comprobar copia de seguridad cuando haya terminado.**
    - **Realizar suma de comprobación antes de escribir en los medios de comunicación.**



6.1.4 Restaurando una copia de seguridad

Un escenario de restauración es un proceso que restaura los datos de una o más copias de seguridad y se recupera la base de datos cuando la última copia de seguridad se restaura.

6.1.4.1 Restore Transact-SQL

Restaura copias de seguridad realizadas con el comando BACKUP. Este comando le permite realizar los siguientes escenarios de restauración:

- Restaurar una base de datos completa a partir de una copia de seguridad completa de la base de datos (restauración completa).
- Restaurar parte de una base de datos (restauración parcial).
- Restaurar archivos o grupos de archivos en una base de datos.
- Restaurar páginas específicas en una base de datos (restauración de páginas).
- Restaurar un registro de transacciones en una base de datos (restauración del registro de transacciones).
- Revertir una base de datos al punto temporal capturado por una instantánea de la base de datos.

La sintaxis para restaurar una base de datos desde una copia de seguridad completa

```
RESTORE DATABASE { nombre_base_datos | @var_nombre_base_datos }
 [ FROM <backup_device> [ ,...n ] ]
 [ WITH
   {
    [ RECOVERY | NORECOVERY | STANDBY =
        {standby_file_name | @standby_file_name_var }
       ]
```

La sintaxis para restaurar archivos específicos o Filegroups de la base de datos

```
RESTORE DATABASE { nombre_base_datos | @var_nombre_base_datos }
   <file_or_filegroup> [ ,...n ]
 [ FROM < dispositivo_backup > [ ,...n ] ]
   WITH
   {
      [ RECOVERY | NORECOVERY ]
      [ , <general_WITH_options> [ ,...n ] ]
   } [ ,...n ]
[;]
```

La sintaxis para restaurar el registro de transacciones o LOG

```
RESTORE LOG { nombre_base_datos | @var_nombre_base_datos }
 [ <file_or_filegroup_or_pages> [ ,...n ] ]
 [ FROM <ba dispositivo_backup ckup_device> [ ,...n ] ]
 [ WITH    {
     [ RECOVERY | NORECOVERY | STANDBY =
        {standby_file_name | @standby_file_name_var }
       ]
```

Argumentos

Argumento	Descripción
DATABASE	Especifica la base de datos de destino. Si se especifica una lista de archivos y grupos de archivos, sólo se restauran esos archivos y grupos de archivos.
LOG	Especifica que sólo se va a aplicar una copia de seguridad de registro de transacciones a esta base de datos. Los registros de transacciones deben aplicarse en orden secuencial. Para aplicar varios registros de transacciones, utilice la opción NORECOVERY en todas las operaciones de restauración.
{ nombre_base_datos | @var_nombre_base_datos }	Es la base de datos para la que se realiza la restauración del registro o de la base de datos completa. Si se proporciona como una variable (@var_nombre_base_datos), este nombre se puede especificar como una constante de cadena (@var_nombre_base_datos **=***database name*) o como una variable de un tipo de datos de cadena de caracteres
<file_or_filegroup> [,...*n*]	Especifica un grupo de archivos o un archivo de que se van a incluir en una instrucción RESTORE DATABASE o RESTORE LOG. Puede especificar una lista de archivos o grupos de archivos.

FROM <dispositivo_backup> [,...n]	Especifica los dispositivos de copia de seguridad desde los que se restaurará la copia de seguridad. Alternativamente, en una instrucción RESTORE DATABASE, la cláusula FROM puede especificar el nombre de una instantánea de base de datos a la que va a revertir la base de datos, en cuyo caso no se admite ninguna cláusula WITH. Si se omite la cláusula FROM, no se produce la restauración de la copia de seguridad. En su lugar, se recupera la base de datos. Esto permite recuperar una base de datos restaurada con la opción NORECOVERY o cambiar a un servidor en espera. . < dispositivo_backup > Especifica el dispositivo de copia de seguridad físico o lógico que se va a utilizar para la operación de copia de seguridad. { device_logico | @var_device_logico } Es el nombre lógico del dispositivo de copia de seguridad en que se hace la copia de seguridad de la base de datos. El nombre lógico debe seguir las reglas definidas para los identificadores. { DISK | TAPE } **=** { 'device_fisico' | @var_device_fisico } Especifica un archivo de disco o un dispositivo de cinta.
DATABASE_SNAPSHOT **=***database_snapshot_name*	Revierte la base de datos a la instantánea de base de datos especificada por *database_snapshot_name*. La opción DATABASE_SNAPSHOT solo está disponible para una restauración de base de datos completa. En una operación de reversión, la instantánea de base de datos

	ocupa el lugar de una copia de seguridad de base de datos completa. En una operación de reversión, se requiere que la instantánea de base de datos especificada sea la única en la base de datos. Durante la operación de reversión, la instantánea de base de datos y la base de datos de destino se marcan como In restore. .

Opciones de WITH

Opción	**Descripción**
PARTIAL	Especifica una operación de restauración parcial que solo restaura el grupo de archivos principal y cualquiera de los grupos de archivos secundarios especificados. La opción PARTIAL selecciona implícitamente el grupo de archivos principal; por tanto, no es necesario especificar FILEGROUP = 'PRIMARY'. Para restaurar un grupo de archivos secundarios, debe especificarlo de forma explícita mediante la opción FILE o FILEGROUP. La opción PARTIAL no se permite en las instrucciones RESTORE LOG.
[RECOVERY | NORECOVERY | STANDBY]	RECOVERY Indica a la operación de restauración que revierta las transacciones no confirmadas. Después del proceso de recuperación, la base de datos está preparada para ser utilizada, la opción predeterminada es RECOVERY. NORECOVERY Indica a la operación de restauración que no revierta las transacciones no confirmadas. Si se utiliza la opción NORECOVERY durante una operación de restauración sin conexión, la base de datos no puede utilizarse. STANDBY **=***standby_file_name*

	Especifica un archivo en espera que permite deshacer los efectos de la recuperación. La opción STANDBY se puede utilizar en operaciones de restauración sin conexión (incluida la restauración parcial).
{NO_CHECKSUM|CHECKSUM}	NO_CHECKSUM Deshabilita de forma explícita la generación de sumas de comprobación de copia de seguridad (y la validación de sumas de comprobación de página). Es el comportamiento predeterminado, salvo para una copia de seguridad comprimida. CHECKSUM Habilita las sumas de comprobación de copia de seguridad.

El siguiente ejemplo se restaura una copia de seguridad completa de la base de datos desde un dispositivo lógico de copia de seguridad de la base de datos Negocios2011Back

```
RESTORE DATABASE Negocios2011
   FROM Negocios2011Backups
Go
```

En el siguiente ejemplo, se restaura una copia de seguridad completa después de una copia de seguridad diferencial del dispositivo de copia de seguridad D:\SQLNegocios2011.Bak, que contiene las dos copias de seguridad. La copia de seguridad de bases de datos completa que se va a restaurar es el sexto conjunto de copias de seguridad del dispositivo (FILE = 6), y la copia de seguridad de base de datos diferencial es el noveno conjunto del dispositivo (FILE = 9).

```
RESTORE DATABASE NEGOCIOS2011
   FROM NEGOCIOS2011BACKUPS

 RESTORE DATABASE NEGOCIOS2011
   FROM DISK = 'D:\NEGOCIOSDATA.BAK'
   WITH FILE = 6, NORECOVERY;
GO
```

```
RESTORE DATABASE NEGOCIOS2011
   FROM DISK = 'D:\NEGOCIOSDATA.BAK'
   WITH FILE = 9, RECOVERY;
GO
```

En el ejemplo siguiente, se restaura una base de datos completa y el registro de transacciones, y se mueve la base de datos restaurada al directorio C:\Data.

```
RESTORE DATABASE Negocios2011
   FROM Negocios2011Backups
   WITH NORECOVERY,
      MOVE 'Negocios2011_Data' TO 'C:\Data\Negocios2011.mdf',
      MOVE ' Negocios2011_Log' TO 'C:\Data\Negocios2011.ldf'
RESTORE LOG Negocios2011
   FROM Negocios2011Backups
   WITH RECOVERY
```

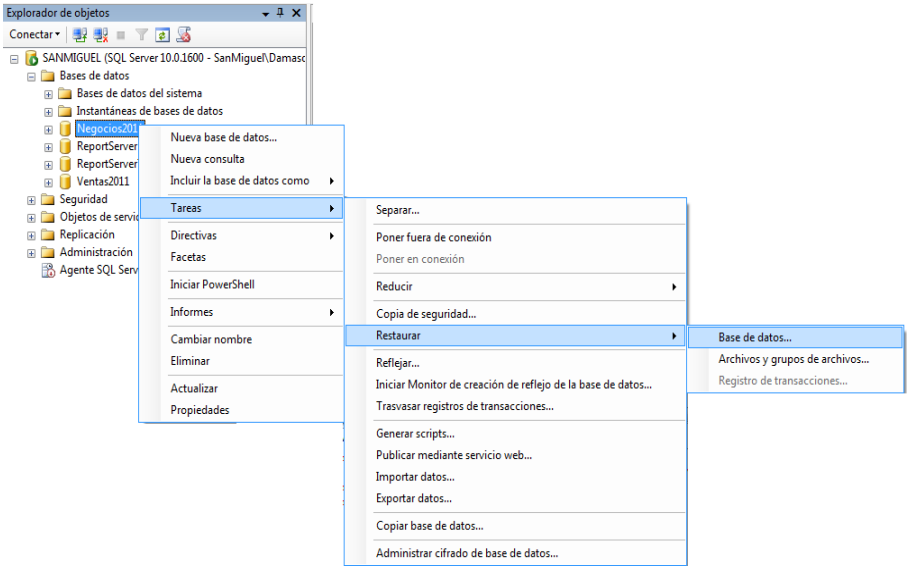
En el ejemplo siguiente, se restaura el registro de transacciones hasta la marca de la transacción marcada denominada ActualizarPrecios.

```
USE NEGOCIOS2011;
GO

BEGIN TRANSACTION ACTUALIZARPRECIOS
   WITH MARK 'UPDATE LISTA PRECIOS';
GO

UPDATE COMPRAS.PRODUCTOS
   SET PRECIOUNIDAD *= 1.10    WHERE NOMPRODUCTO LIKE 'BK-%';
GO

COMMIT TRANSACTION ACTUALIZARPRECIOS;
GO
```

```
USE MASTER
GO

RESTORE DATABASE NEGOCIOS2011
FROM NEGOCIOS2011BACKUPS WITH FILE = 3, NORECOVERY;
GO

RESTORE LOG NEGOCIOS2011
   FROM NEGOCIOS2011BACKUPS  WITH FILE = 4, RECOVERY,
   STOPATMARK = 'ACTUALIZARPRECIOS';
```

6.1.4.2 Restaurando una base de datos utilizando SQL SERVER Management Studio

1. En el Explorador de objetos, expandir el árbol del servidor.
2. Expanda **Bases de datos.** Dependiendo de la base de datos, seleccione una base de datos de usuario.
3. Haga clic derecho en la base de datos, seleccione **Tareas y,** a continuación, haga clic en **Restaurar.**
4. Haga clic en **la base de datos,** que se abre el cuadro de diálogo **Restaurar base de datos.**



Para especificar el origen y la ubicación de la copia de seguridad conjuntos para restaurar, haga clic en una de las siguientes opciones:

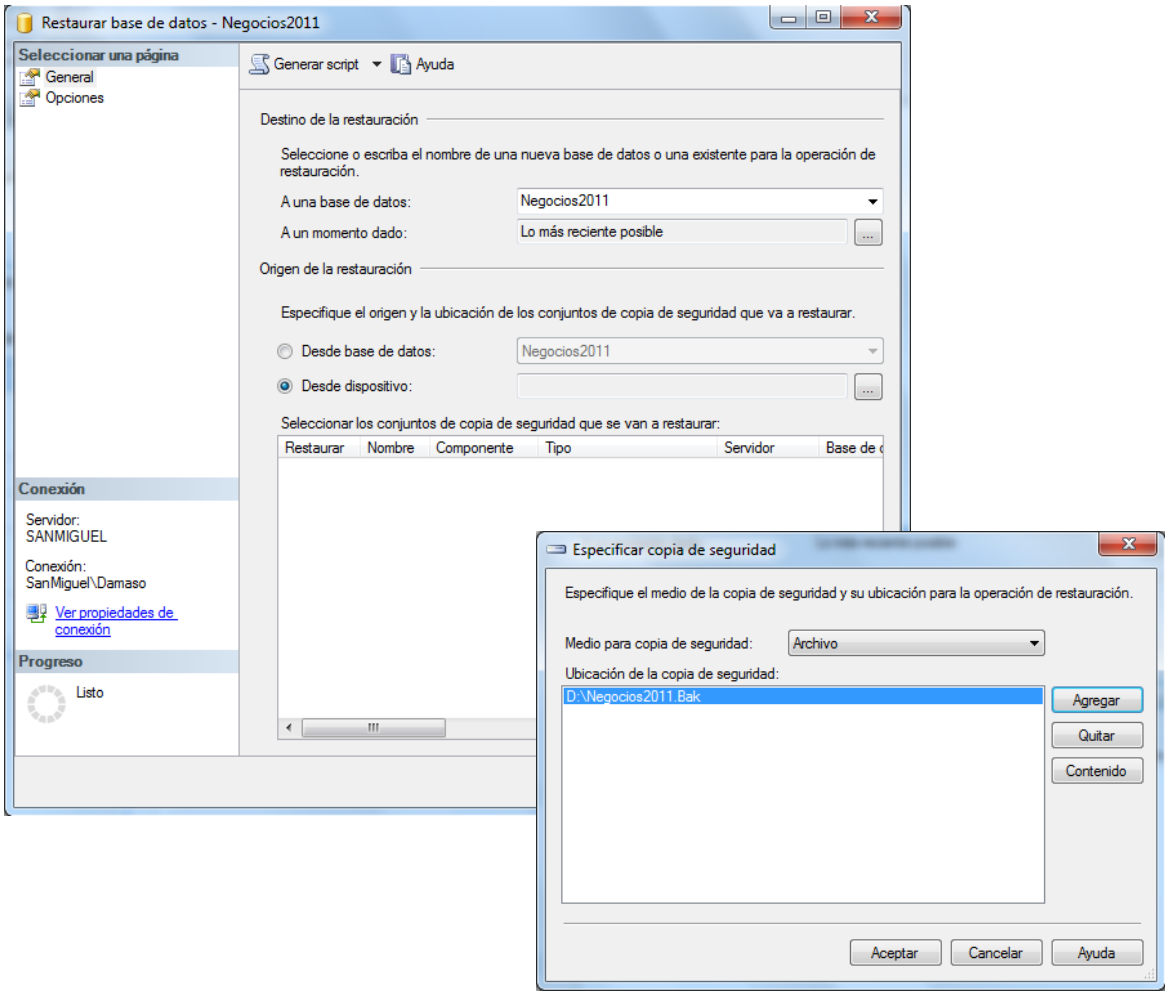
- **Base de datos**

Escriba un nombre de base de datos en el cuadro de lista.

- **Desde el dispositivo**

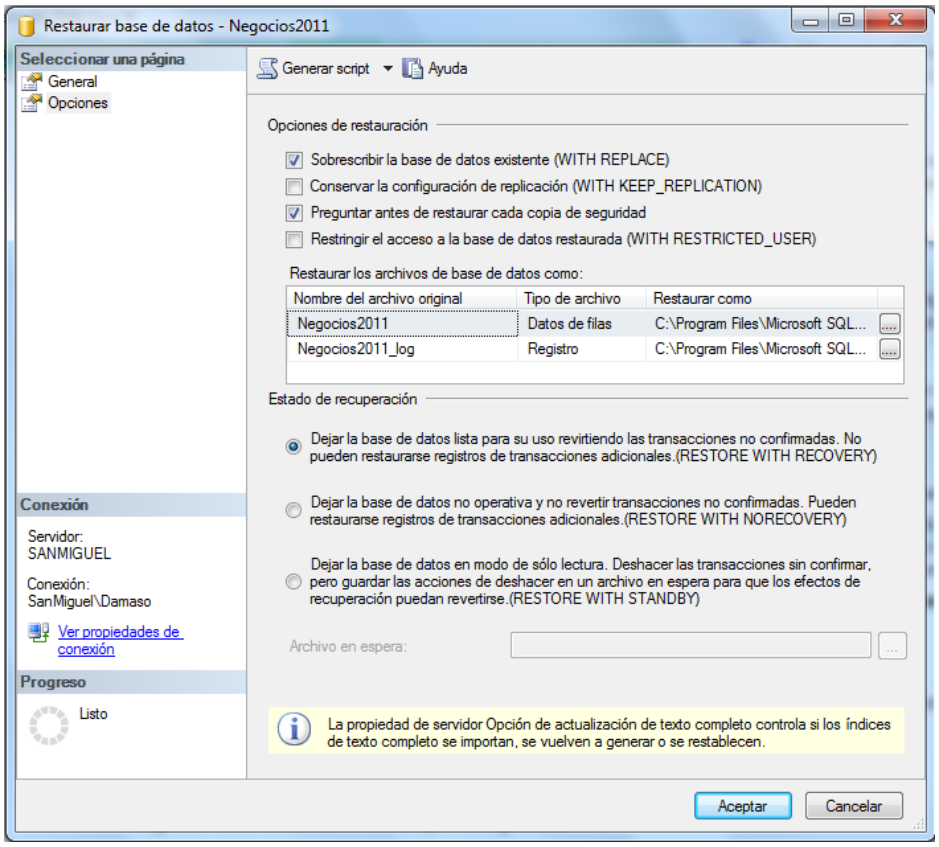
Haga clic en el botón Examinar, que abrirá el cuadro de diálogo **Especificar copia de seguridad.** En el cuadro de lista **Copia de seguridad de los medios de comunicación,** seleccione uno de los tipos de dispositivo. Para seleccionar uno o varios dispositivos del cuadro de lista **Ubicación de copia de seguridad,** haga clic en **Agregar.**

Después de agregar los dispositivos que desee al cuadro de lista **Ubicación de copia de seguridad,** haga clic en **Aceptar** para regresar a la página **General.**



En el panel **Opciones de restauración,** puede elegir cualquiera de las siguientes opciones, si es apropiado para su situación:

1. **Sobrescribir la base de datos existente**
2. **Conservar la configuración de la replicación**
3. **Preguntar antes de restaurar cada copia de seguridad**
4. **Restringir el acceso a la base de datos restaurada**



Resumen

- El propósito de crear copias de seguridad de SQL Server es para que usted pueda recuperar una base de datos dañada. Sin embargo, copias de seguridad y restauración de los datos deben ser personalizados para un ambiente particular y debe trabajar con los recursos disponibles. Por lo tanto, un uso fiable de copia de seguridad y restauración para la recuperación exige una copia de seguridad y restauración de la estrategia.
- La mejor opción de modelo de recuperación de la base de datos depende de los requerimientos de su negocio. Para evitar la gestión del registro de transacciones y simplificar el BACKUP y restauración, utilice el modelo de recuperación simple. Para minimizar la pérdida de trabajo, a costa de los gastos generales de administración, utilice el modelo de recuperación completa
- El alcance de una copia de seguridad de los datos puede ser una base de datos completa, una base de datos parciales, o un conjunto de archivos o grupos de archivos. Para cada uno de estos, SQL Server admite copias de seguridad completas y diferenciales
- Bajo el modelo de recuperación optimizado para cargas masivas de registros de modelo de recuperación, *copias de seguridad del registro de transacciones* (o *copias de seguridad de registro)* son obligatorias. Cada copia de seguridad de registro cubre la parte del registro de transacciones que estaba activa cuando la copia de seguridad fue creada, e incluye todos los registros que no fueron respaldados en una copia de seguridad de registros anterior
- BACK UP TRANSACT-SQL realiza copias de seguridad de una base de datos completa, o uno o más archivos o grupos de archivos (BASE DE DATOS DE SEGURIDAD). Además, bajo el modelo de recuperación optimizado para cargas masivas de registros de modelo de recuperación, copias de seguridad del registro de transacciones (BACKUP LOG).
- Un escenario de restauración es un proceso que restaura los datos de una o más copias de seguridad y se recupera la base de datos cuando la última copia de seguridad se restaura

📖 Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.

- **¡Error! Referencia de hipervínculo no válida.** http://technet.microsoft.com/es-es/library/ms186858.aspx

  Aquí hallará los conceptos RESTORE TRANSACT SQL

- **¡Error! Referencia de hipervínculo no válida.**http://technet.microsoft.com/es-es/library/ms186865.aspx

  En esta página, hallará los conceptos de BACK UP TRANSACT SQL

- http://translate.googleusercontent.com/translate_c?hl=es&prev=/search%3Fq%3DPlanning%2Ba%2Bbackup%2Bstrategy%2Bin%2Bsql%2Bserver%26hl%3Des%26biw%3D1280%26bih%3D619%26prmd%3Divns&rurl=translate.google.com.pe&sl=en&u=http://msdn.microsoft.com/en-us/library/ms187048.aspx&usg=ALkJrhgYgMJDrnvutDfU9eIJ2B0a0Ps1mw

  Aquí hallará los conceptos de copia de seguridad y recuperación.