



BASE DE DATOS AVANZADO I (LABORATORIO)

Índice

Presentación	7
Red de contenidos	9
Unidad de Aprendizaje 1	
BASE DE DATOS Y TABLAS	
1.1 Tema 1 : Manejo de una Base de Datos	11
1.1.1 : Definición de una base de datos	11
1.1.2 : Creación de una base de datos	13
1.1.3 : Creación de Filegroup	16
1.1.4 : Actualización de archivos de una base de datos	17
1.1.5 : Agregar archivos secundarios a una base de datos	18
1.2 Tema 2 : Manejo de tablas y esquemas	20
1.2.1 : Creación de esquemas	20
1.2.2 : Creación de tipos de datos de usuario	22
1.2.3 : Creación de tablas	23
1.2.4 : Modificación de una estructura de una tabla	26
1.2.5 : Manejo de una tabla particionada	27
1.3. Tema 3 : Manejo de restricciones e índices	30
1.3.1 : Constraints o restricciones (tipos)	30
1.3.2 : Uso del Identity	33
1.3.3 : Concepto de índices, tipos de índices	35
1.3.4 : Índices particionados	38
Unidad de Aprendizaje 2	
LENGUAJE DE MANIPULACIÓN DE DATOS	41
2.1 Tema 4 : Lenguaje de manipulación de datos – DML	42
2.1.1 : Inserción de datos INSERT y BULK INSERT	42
2.1.2 : Actualización de datos UPDATE	47
2.1.3 : Eliminación de datos DELETE	48
2.1.4 : Declaración MERGE	49
2.2 Tema 5 : Recuperación de datos	52
2.2.1 : Consulta de datos, uso del SELECT	52
2.2.2 : Ordenar registros	54
2.2.3 : Consultas condicionales, uso de operadores condicionales	54
2.2.4 : Empleo de funciones agregadas: SUM, MIN, MAX, AVG, COUNT	57
2.2.5 : Uso de las cláusulas GROUP BY y HAVING	60
2.2.5.1 : Opciones CUBE y ROLLUP	62
2.3 Tema 6 : Recuperación de datos II	69
2.3.1 : Combinación de tablas	69
2.3.2 : Combinaciones Internas: INNER JOIN	69
2.3.3 : Combinaciones Externas: LEFT JOIN, RIGHT JOIN	69
2.3.4 : Agregar conjunto de resultados: UNION	71

Unidad de Aprendizaje 3**INTRODUCCIÓN A LA PROGRAMACIÓN TRANSACT SQL** 76**3.1 Tema 7 : Sentencias SQL para la programación** 77

- 3.1.1 : Fundamentos de la programación con Transact SQL 77
- 3.1.2 : Identificadores 77
- 3.1.3 : Variables: declaración, asignación 78
- 3.1.4 : Elementos de flujo de control 79
- 3.1.4.1 : Estructuras de control IF 80
- 3.1.4.2 : Estructuras de control CASE 82
- 3.1.4.3 : Estructuras de control WHILE 84
- 3.1.5 : Control de errores con TRY / CATCH, uso de @@ERROR, uso de RAISERROR 86
- 3.1.6 : Uso detransacción: COMMIT Y ROLLBACK 90

Unidad de Aprendizaje 4**CREACIÓN DE CURSORES** 95**4.1 Tema 8 : Construcción de Cursores** 91

- 4.1.1 : Construcción de cursores explícitos e implícitos 97
- 4.1.2 : Cursores de actualización de datos 104

Unidad de Aprendizaje 5**PROGRAMACIÓN TRANSACT SQL** 107**5.1 Tema 9 : Manejo de procedimientos almacenados** 108

- 5.1.1 : Definición y tipos 108
- 5.1.2 : Construcción de procedimientos almacenados 110
- 5.1.3 : Manejo de parámetros: valores de entrada y valores de retorno 113

Manejo de procedimientos almacenados II**5.2 Tema 10 : Anidamiento con procedimientos almacenados** 115

- 5.2.1 : Operaciones con procedimientos almacenados 115
- 5.2.2 : **Manejo de funciones de usuario** 117

5.3 Tema 11 : Funciones del sistema 124

- 5.3.1 : Funciones de usuario 124
- 5.3.2 : Funciones escalares 129
- 5.3.2.1 : Funciones de tabla en línea 129
- 5.3.2.2 : Funciones multisentencias 131
- 5.3.2.3 : 132

5.4 Tema 12 : Desencadenadores 136

- 5.4.1 : Desencadenadores DML 136
- 5.4.2 : Desencadenadores DDL 144

Unidad de Aprendizaje 6**ADMINISTRACION DE BASE DE DATOS EN SQL SERVER** 150**6.1 Tema 13 : Seguridad y restauración en SQL Server** 151

- 6.1.1 : Copia de seguridad en SQL Server 152
- 6.1.1.1 : Tipos de Backup 154
- 6.1.2 : Restaurando una copia de seguridad 161
- 6.1.3 : Base de datos SnapShots 169

6.1.3.1 : Creando DB SnapShot	169
6.1.3.2 : Utilizar un SnapShot para revertir cambios	171
6.2 Tema 14 : Automatizar tareas en SQL	172
6.2.1 : Creando tareas	172
6.2.2 : Creando alertas	180
6.2.3 : Creando planes de mantenimiento	183
 Unidad de Aprendizaje 7	
MANEJO DE DATOS EN XML	187
7.1 Tema 15 : Base de dato relacionales para datos XML	188
7.1.1 : Introducción	188
7.1.2 : Tipos de datos XML	188
7.1.3 : FOR XML y mejoras XML	192
7.2 Tema 16 : Procesamiento XML en SQL Server	194
7.2.1 : Almacenamiento de datos XML	194
7.2.2 : Recuperación de datos de tipo XML	196
7.2.2.1 : Usar modo RAW	196
7.2.2.2 : Usar modo AUTO	199
7.2.2.3 : Usar modo EXPLICIT	202

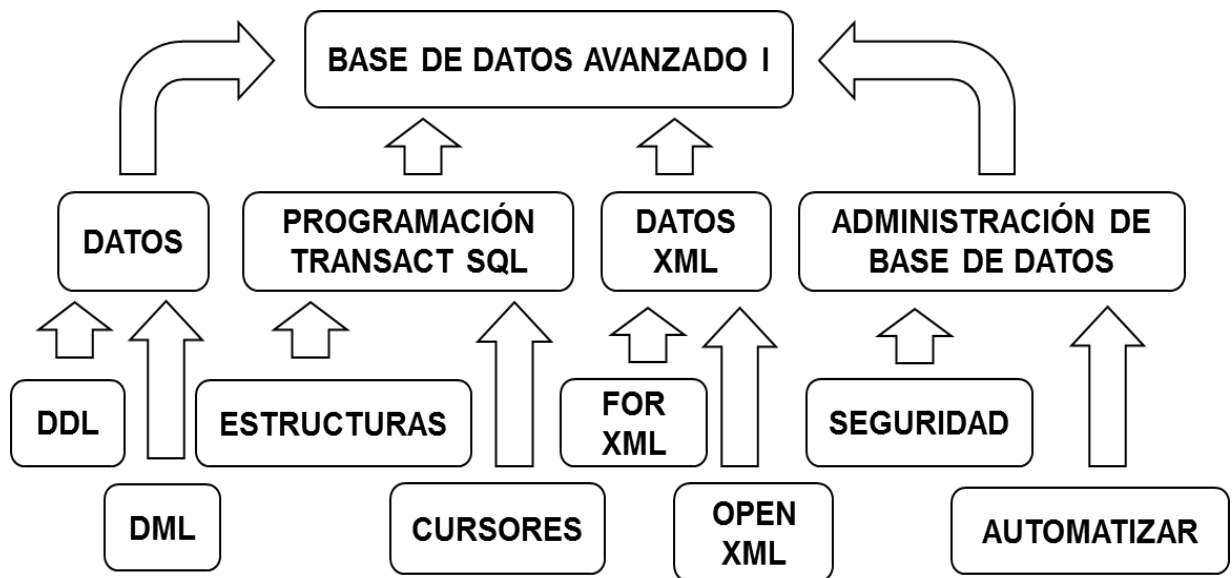
Presentación

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 optimizar una base de datos logrando alta disponibilidad, seguridad y rendimiento, todo ello utilizando comandos TRANSACT-SQL.

El manual 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 2014. 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 cláusulas 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.

Se recomienda la lectura del material de la unidad correspondiente antes de asistir a clase, con el fin de fomentar la discusión del tema y facilitar la comprensión del mismo.

Red de contenidos





BASE DE DATOS Y TABLAS

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno crea y maneja una base de datos y sus esquemas; además crea e implementan tablas de base de datos para almacenar los datos de un proceso de negocio real, así como manejar índices e implementarlos en las tablas o vistas de una base de datos.

TEMARIO

1.1 Tema 1 : Manejo de una Base de Datos

- 1.1.1 : Definición de una base de datos
- 1.1.2 : Creación de una base de datos
- 1.1.3 : Creación de Filegroup
- 1.1.4 : Actualización de archivos de una base de datos
- 1.1.5 : Agregar archivos secundarios a una base de dato

1.2 Tema 2 : Manejo de tablas y esquemas

- 1.2.1 : Creación de esquemas
- 1.2.2 : Creación de tipos de datos de usuario
- 1.2.3 : Creación de una tabla
- 1.2.4 : Modificación de la estructura de una tabla
- 1.2.5 : Manejo de una tabla particionada

1.3 Tema 3 : Manejo de restricciones e índices

- 1.3.1 : Constraints o restricciones Tipos.
- 1.3.2 : Uso de Identity
- 1.3.3 : Concepto de Índices, tipos de índices.
- 1.3.4 : Índice particionado

ACTIVIDADES PROPUESTAS

- Los alumnos identifican las opciones de una base de datos, tablas e índices.
- Los alumnos diseñan e implementan una base de datos optimizada
- Los alumnos diseñan e implementan tablas, esquemas e índices.

1.1. MANEJO DE UNA BASE DE DATOS

1.1.1. Definición de una base de datos

Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido, una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos están en formato digital (electrónico), y por ende se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos. Existen programas denominados sistemas gestores de bases de datos (SGBD), que permiten almacenar y posteriormente acceder a los datos de forma rápida y estructurada. Las propiedades de estos SGBD, así como su utilización y administración, se estudian dentro del ámbito de la informática.

Las aplicaciones más usuales son para la gestión de empresas e instituciones públicas. También son ampliamente utilizadas en entornos científicos con el objeto de almacenar la información experimental.

Microsoft SQL Server es un sistema para la gestión de bases de datos producido por Microsoft basado en el modelo relacional. Sus lenguajes para consultas son T-SQL y ANSI SQL. Microsoft SQL Server constituye la alternativa de Microsoft a otros potentes sistemas gestores de bases de datos como son Oracle, PostgreSQL o MySQL.

SQL Server incluye las siguientes bases de datos del sistema.

Base de datos del sistema	Descripción
master	Registros de toda la información a nivel de sistema para una instancia de SQL Server.
msdb	Es utilizado por el Agente SQL Server para programar alertas y puestos de trabajo.
model	Se utiliza como plantilla para todas las bases de datos creadas en la instancia de SQL Server. Las modificaciones introducidas en el modelo de base de datos, tales como el tamaño de la base de datos, el cotejo, modelo de recuperación, y otras opciones de base de datos, se aplican a las bases de datos creadas después.
Resource (SQL Server 2016)	Es una base de datos de sólo lectura que contiene los objetos del sistema que se incluyen con SQL Server. Los objetos del sistema se conservan físicamente en el de recursos de base de datos, pero lógicamente aparecen en el SYS esquema de cada base de datos.
tempdb	Es un espacio de trabajo para la celebración de objetos temporales o conjuntos de resultados intermedios.

1.1.2. Creación de una base de datos

Para crear una base de datos en SQL Server, se ejecuta la instrucción CREATE DATABASE en modo de confirmación automática (modo predeterminado de administración de transacciones) y no se permite en una transacción explícita o implícita.

Cada vez que se crea, modifica o quita una base de datos de usuario, se debe hacer una copia de seguridad de la base de datos master.

Cuando cree una base de datos, defina el máximo tamaño posible para los archivos de datos según la cantidad de datos máxima prevista para la base de datos.

En una instancia de SQL Server se pueden especificar 32,767 base de datos como máximo.

Sintaxis:

```
CREATE DATABASE database_name
[ ON
    [ PRIMARY ] <filespec> [ ,...n ]
    [ , <filegroup> [ ,...n ] ]
    [ LOG ON <filespec> [ ,...n ] ]
]
```

Para adjuntar una base de datos

CREATE DATABASE database_name

```
ON <filespec> [ ,...n ]
FOR { { ATTACH [ WITH <attach_database_option> [ , ...n ] ] }
    | ATTACH_REBUILD_LOG }
[:]

<filespec> ::=
{
(
    NAME = logical_file_name ,
    FILENAME = { 'os_file_name' | 'filestream_path' }
    [ , SIZE = size [ KB | MB | GB | TB ] ]
    [ , MAXSIZE = { max_size [ KB | MB | GB | TB ] | UNLIMITED } ]
    [ , FILEGROWTH = growth_increment [ KB | MB | GB | TB | % ]
]
)
}

<filegroup> ::=
{
FILEGROUP filegroup_name [ CONTAINS FILESTREAM ] [ DEFAULT ]
    <filespec> [ ,...n ]
}
```

```
<attach_database_option> ::=  
{  
    <service_broker_option>  
    | RESTRICTED_USER  
    | FILESTREAM ( DIRECTORY_NAME = { 'directory_name' | NULL }  
)  
}
```

Crear una base de datos snapshot

```
CREATE DATABASE database_snapshot_name ON  
(  
    NAME = logical_file_name,  
    FILENAME = 'os_file_name'  
) [ ,...n ]  
AS SNAPSHOT OF source_database_name  
[:]
```

Caso Práctico: Creando una base de datos Comercial

```
/*CREAR LA BASE DE DATOS COMERCIAL*/  
CREATE DATABASE COMERCIAL  
GO  
  
/*ACTIVAR LA BASE DE DATOS CREADA*/  
USE COMERCIAL  
GO
```

Caso Práctico: Creando una base de datos CiberData definiendo cada uno de sus archivos

```
/*Implemente una Base de datos llamada CiberData el cual está formada por dos
archivos:

    • El archivo de datos: CiberData cuya ubicación se encontrará en el disco duro; su
      tamaño inicial es de 10MB y su máximo es ilimitado.

    • El archivo de transacciones: CiberDataLog cuya ubicación se encontrará en el
      disco duro; su tamaño máximo es de 500MB y factor de crecimiento es de 5MB
*/

CREATE DATABASE CIBERDATA
ON PRIMARY (
    NAME=CIBERDATA,
    FILENAME='M:\CIBERDATA.MDF',
    SIZE=10MB,
    MAXSIZE=UNLIMITED
)
LOG ON (
    NAME=CIBERDATALOG,
    FILENAME='M:\CIBERDATALOG.LDF',
    MAXSIZE=500MB,
    FILEGROWTH=5MB
)
GO

/*Ver la estructura de la base de datos*/
SP_HELPDB CIBERDATA
GO

/*Activar la base de datos*/
USE CIBERDATA
GO
```

Para adjuntar o acoplar una base de datos, se debe tener en cuenta lo siguiente:

- La base de datos se debe separar primero. Si intenta adjuntar una base de datos que no se ha separado, se devolverá un error.
- Al adjuntar una base de datos, todos los archivos de datos deben estar disponibles (archivos MDF y LDF). Si algún archivo de datos tiene una ruta de acceso diferente a la que tenía cuando se creó la base de datos o cuando ésta se adjuntó por última vez, debe especificar la ruta actual.
- Cuando se adjunta una base de datos, si los archivos MDF y LDF se encuentran en directorios diferentes y una de las rutas de acceso incluye \\?\GlobalRoot, se producirá un error en la operación.

Se recomienda no adjuntar ni restaurar bases de datos de orígenes desconocidos o que no sean de confianza. Es posible que dichas bases de datos contengan código

malintencionado que podría ejecutar código Transact-SQL no deseado o provocar errores al modificar el esquema o la estructura de la base de datos física. Para usar una base de datos desde un origen desconocido o que no sea de confianza, ejecute DBCC CHECKDB.

Caso Práctico: Adjuntando una base de datos NewCiberData indicando los filename.

```
CREATE DATABASE NEWCIBERDATA
ON
    (FILENAME='M:\CIBERDATA.MDF')
LOG ON
    (FILENAME='M:\CIBERDATALOG.LDF')
FOR ATTACH
GO
```

1.1.3. Creación de Filegroup

Los FileGroups es un concepto muy importante sobre todo cuando vamos a crear una base de datos ya que nos va ayudar mejorar el rendimiento de las consultas y la administración de la data en una BD.

Como concepto FileGroup es una unidad lógica que almacena archivos físicos que pueden estar en distintas unidades de disco o en distintos discos físicos, la idea del Filegroup es la de aprovechar el procesamiento paralelo cuando el motor de BD requiere realizar operaciones de I/O al disco duro.

Tener en cuenta que cuando creamos una Base de Datos, el SQL crea un FileGroup por defecto: ON PRIMARY, en este FileGroup se almacenan todas las tablas del sistema (si es que no sabías te lo cuento, cada vez que creamos una nueva Base de datos se hace una copia de todas las tablas de la BD Model) y todos los objetos que vayamos creando.

La recomendación general es dejar a las tablas del sistema en el FileGroup por Defecto (el que crea SQL) y crear un Filegroup para los objetos de usuario (las tablas, SP, Trigger, etc creado por nosotros), adicionalmente también podrías crear un Filegroup para almacenar data histórica que sabemos que no cambia en el tiempo.

Por último tener en cuenta que el archivo de transacciones no se asocia a ningún FileGroup (Toda BD consta de un archivo mdf, opcionalmente archivos ndf y uno o más archivos ldf).

Este query asume que UD está familiarizado con los conceptos de archivos mdf, ndf y ldf.

```
/*Agregando FileGroups en la Base de Datos*/
ALTER DATABASE BD_Negocios ADD FILEGROUP DATA_1
ALTER DATABASE BD_Negocios ADD FILEGROUP DATA_2
Go

/*Agregando archivos secundarios a los FileGroups*/
ALTER DATABASE BD_Negocios
ADD FILE (NAME = Data1,
```



```

FILENAME = 'D:\DATA1.ndf', SIZE = 1MB,
MAXSIZE = 10MB, FILEGROWTH = 1MB
) TO FILEGROUP DATA_1

```

Go

```

ALTER DATABASE BD_Negocios
ADD FILE (NAME = Data2,
FILENAME = 'D:\DATA_2.ndf',
SIZE = 1MB, MAXSIZE = 10MB, FILEGROWTH = 1MB
) TO FILEGROUP DATA_2

```

Go

1.1.4. Actualización de archivos de una base de datos

El comando ALTER DATABASE permite modificar los archivos y grupos de archivos asociados con la base de datos. Permite agregar o eliminar los archivos y grupos de archivos de una base de datos, y los cambios de los atributos del archivo de una base de datos.

Sintaxis:

```

ALTER DATABASE
{
<add_or_modify_files>
| <add_or_modify_filegroups>
}
[:]

<add_or_modify_files> :: =
{
ADD FILE <filespec> [, ... n]
[A FILEGROUP {} filegroup_name]
| ADD LOG FILE <filespec> [, ... n]
| REMOVE FILE logical_file_name
| MODIFY <filespec> ARCHIVO
}

<add_or_modify_filegroups> :: =
{
| ADD FILEGROUP filegroup_name
[CONTIENE FILESTREAM]
| REMOVE FILEGROUP filegroup_name
| MODIFY FILEGROUP filegroup_name
{<filegroup_updatability_option>
| DEFAULT
| NOMBRE = new_filegroup_name
}
}

```

```
<filegroup_updatability_option> :: =  
{  
{READONLY | READWRITE}  
| { | READ_ONLY READ_WRITE}  
}
```

Caso Práctico: Modificando un archivo de base de datos CiberData

```
/* El administrador debe ampliar el tamaño inicial del archivo secundario a 10MB y su  
incremento deberá ser 15%*/  
ALTER DATABASE CIBERDATA  
MODIFY FILE (  
    NAME=CIBERDATASEC,  
    SIZE=10MB,  
    FILEGROWTH=15%  
)  
GO
```

Caso Práctico: Modificar la ubicación de los archivos de la base de datos CiberData.

```
/* El administrador debe modificar la ubicación de los archivo primario y log de la base  
de datos CiberData*/  
  
USE MASTER;  
GO  
  
ALTER DATABASE CiberData  
MODIFY FILE (NAME = CiberData,  
FILENAME = 'E: \ SQLData \ CiberData.mdf');  
GO  
  
ALTER DATABASE tempdb  
MODIFY FILE (NAME = CiberDataLog,  
FILENAME = 'E: \ SQLData \ CiberDataLog.ldf');  
GO
```

Caso Práctico: Eliminar un archivo de la base de datos CiberData.

```
/* El administrador debe eliminar el archivo secundario*/  
USE master;  
GO  
  
ALTER DATABASE CIBERDATA  
REMOVE FILE CIBERDATASEC  
GO
```

1.1.5. Agregar archivos de una base de datos

Una base de datos administrada por SQL Server permite agregar archivos secundarios de extensión ndf (archivos que almacenan datos), los cuales permiten almacenar los datos cuando el archivo primario (mdf) se encuentra lleno.

Para definir un archivo secundario se puede hacer de dos formas:

- Creando los archivos desde el comando CREATE DATABASE
- Agregando los archivos secundarios a la base de datos ya creada a través el comando ALTER DATABASE

Caso Práctico: Crear la base de datos CiberData agregando el archivo secundario.

```
CREATE DATABASE CIBERDATA
ON PRIMARY(
    NAME=CIBERDATA,
    FILENAME='D:\CIBERDATA.MDF',    SIZE=10MB,
MAXSIZE=UNLIMITED
),
(
    NAME=CIBERDATASEC,
    FILENAME='D:\CIBERDATASEC.NDF',  SIZE=10MB,
MAXSIZE=UNLIMITED
)
LOG ON(
    NAME=CIBERDATALOG,
    FILENAME='D:\CIBERDATALOG.LDF',  MAXSIZE=500MB,
FILEGROWTH=5MB
)
GO
```

Caso Práctico: Agregar un archivo secundario a la base de datos CiberData

/*El administrador evalúa que su base de datos se llenara pronto por lo que debe agregar un archivo secundario llamado CiberDataSec cuyo tamaño inicial será de 5MB, su tamaño máximo es de 500MB y su crecimiento es de 10%*/

```
ALTER DATABASE CIBERDATA
ADD FILE (
    NAME=CIBERDATASEC,
    FILENAME='M:\CIBERDATASEC.NDF',
    SIZE=5MB,
    MAXSIZE=500MB,
    FILEGROWTH=10%
)
GO
```

Caso Práctico: Eliminar un archivo secundario a la base de datos CiberData

/*El administrador debe eliminar el archivo secundario*/

```
ALTER DATABASE CIBERDATA
REMOVE FILE CIBERDATASEC
GO
```

1.2. MANEJO DE TABLAS Y ESQUEMAS

Una tabla es un objeto de base de datos, utilizada para organizar y presentar información.

Las tablas se componen de dos estructuras:

- Registro: es cada una de las filas en que se divide la tabla. Cada registro contiene datos de los mismos tipos que los demás registros. Ejemplo: en una tabla de nombres y direcciones, cada fila contendrá un nombre y una dirección.
- Campo: es cada una de las columnas que forman la tabla. Contienen datos de tipo diferente a los de otros campos. En el ejemplo anterior, un campo contendrá un tipo de datos único, como una dirección, o un número de teléfono, un nombre, etc.

A los campos se les puede asignar, además, propiedades especiales que afectan a los registros insertados. El campo puede ser definido como índice o autoincrementable, lo cual permite que los datos de ese campo cambien solos o sean el principal a la hora de ordenar los datos contenidos.

Cada tabla creada debe tener un nombre único en la Base de Datos, haciéndola accesible mediante su nombre o su seudónimo (Alias) (dependiendo del tipo de base de datos elegida). La estructura de las tablas viene dado por la forma de un archivo plano, los cuales en un inicio se componían de un modo similar.

1.2.1 Creación de esquemas

Un esquema es un contenedor de objetos. Cada esquema es un espacio de nombres distinto que existe de forma independientemente del usuario de base de datos que lo creó. Cualquier usuario puede ser propietario de un esquema, y esta propiedad es transferible.

A partir de SQL Server 2005, los esquemas son entidades explícitas reflejadas en los metadatos y, como resultado, los esquemas solo pueden tener un propietario. Sin embargo, un único usuario puede poseer muchos esquemas.

Los esquemas te sirven sobre todo para organizar los objetos, y también la seguridad de la base de datos, especialmente cuando tengas varios usuarios. Si un usuario posee un esquema, tiene por defecto permisos sobre los objetos de ese esquema que no tendrán otros usuarios, a menos que se les hayan concedido expresamente.

El esquema predeterminado es el primer esquema que se busca al resolver los nombres de objeto no calificadas. Si no hay ningún esquema predeterminado se define para una cuenta de usuario, SQL Server asumirá dbo.

Sintaxis:

CREATE SCHEMA schema_name_clause [<schema_element> [...n]]

```
<schema_name_clause> ::=  
{  
  schema_name  
  | AUTHORIZATION owner_name  
  | schema_name AUTHORIZATION owner_name  
}  
  
<schema_element> ::=  
{  
  table_definition | view_definition | grant_statement |  
  revoke_statement | deny_statement  
}
```

Argumentos:

schema_name

Es el nombre por el que se identifica al esquema en esta base de datos.

AUTHORIZATION owner_name

Especifica el nombre de la entidad de seguridad de la base de datos que poseerá el esquema. Es posible que esta entidad de seguridad posea otros esquemas y no utilice el esquema actual como predeterminado.

table_definition

Especifica una instrucción CREATE TABLE que crea una tabla en el esquema. La entidad de seguridad que ejecuta esta instrucción debe tener el permiso CREATE TABLE en la base de datos actual.

view_definition

Especifica una instrucción CREATE VIEW que crea una vista en el esquema. La entidad de seguridad que ejecuta esta instrucción debe tener el permiso CREATE VIEW en la base de datos actual.

grant_statement

Especifica una instrucción GRANT que otorga permisos sobre cualquier elemento protegible, excepto el esquema nuevo.

revoke_statement

Especifica una instrucción REVOKE que revoca permisos sobre cualquier elemento protegible, excepto el esquema nuevo.

deny_statement

Especifica una instrucción DENY que deniega permisos sobre cualquier elemento protegible, excepto el esquema nuevo.

Caso Práctico: Crear en la base de datos CIBERDATA el esquema RRHH

```
--ACTIVA LA BASE DE DATOS  
USE CIBERDATA  
GO
```

```
--CREA EL SCHEMA RRHH  
CREATE SCHEMA RRHH  
GO  
  
--VER LOS SCHEMAS DE LA BASE DE DATOS  
SELECT * FROM SYS.SCHEMAS  
GO
```

1.2.2 Creación de tipos de datos de usuario

Los tipos de datos definidos por el usuario, permiten afinar aún más los tipos de datos para garantizar la coherencia cuando se trabaja con elementos de datos comunes en diferentes tablas o bases de datos.

Los tipos de datos definidos por el usuario no permiten definir estructuras o tipos de datos complejos. Se define para una base de datos específica.

Existen 02 formas para crear tipos de datos de usuario, con el procedimiento del sistema SP_ADDTYPE o con el comando CREATE TYPE.

Con el procedimiento SP_ADDTYPE.

Sintaxis:

```
sp_addtype [typename =] tipo,  
    [phystype =] System_data_type  
    [, [nulltype =] 'Null_type'];
```

Argumentos:

[**typename** =] *tipo*

Es el nombre del tipo de dato. Los nombres de tipos de datos deben seguir las reglas de los identificadores y deben ser únicos en cada base de datos. *Tipo* es **nombre_sist**, no tiene valor predeterminado.

[**phystype** =] *System_data_type*

Es el tipo de datos del sistema en que se basa el tipo de datos de usuario.

Caso Práctico: Crear en la base de datos master, el tipo de dato SSN basado en un tipo varchar.

```
USE master;  
GO  
  
EXEC sp_addtype ssn, 'varchar (11)', 'NOT NULL';  
GO
```

Caso Práctico: Crear un tipo de dato (basado en DateTime) con nombre de birthday que permite valores nulos.

```
USE master;  
GO  
  
EXEC sp_addtype birthday, 'datetime', 'NULL';  
GO
```

Caso Práctico: Crear tipos de datos telephone y fax.

```
USE master;  
GO  
  
EXEC sp_addtype telephone, 'varchar (24)', 'NOT NULL';  
GO  
  
EXEC sp_addtype fax, 'varchar (24)', 'NULL';  
GO
```

Caso Práctico: Eliminar el tipo de dato BirthDay

```
USE master;  
GO  
  
EXEC sp_droptype 'birthday';  
GO
```

Con el comando CREATE TYPE.

Sintaxis:

```
CREATE TYPE [ schema_name. ] type_name  
{  
    FROM base_type  
    [ ( precision [ , scale ] ) ]  
    [ NULL | NOT NULL ]  
    | EXTERNAL NAME assembly_name [ .class_name ]  
    | AS TABLE ( { <column_definition> | <computed_column_definition> }  
        [ <table_constraint> ] [ ,...n ] )  
} [ ; ]
```

Caso Práctico: Crear el tipo de dato SSN en la base de datos CIBERDATA

```
USE CIBERDATA  
GO  
  
CREATE TYPE SSN  
FROM varchar (11) NOT NULL;  
GO
```

Caso Práctico: Eliminar el tipo de dato SSN.

```
DROP TYPE SSN;  
GO
```

1.2.3 Creación de una tabla

Una tabla es una herramienta de organización de información que se utiliza en bases de datos.

En computación, una tabla hace referencia al modelado o recopilación de datos por parte de una aplicación de un programa que permite operar con los mismos organizándolos y poniéndolos en relación de diversas maneras.

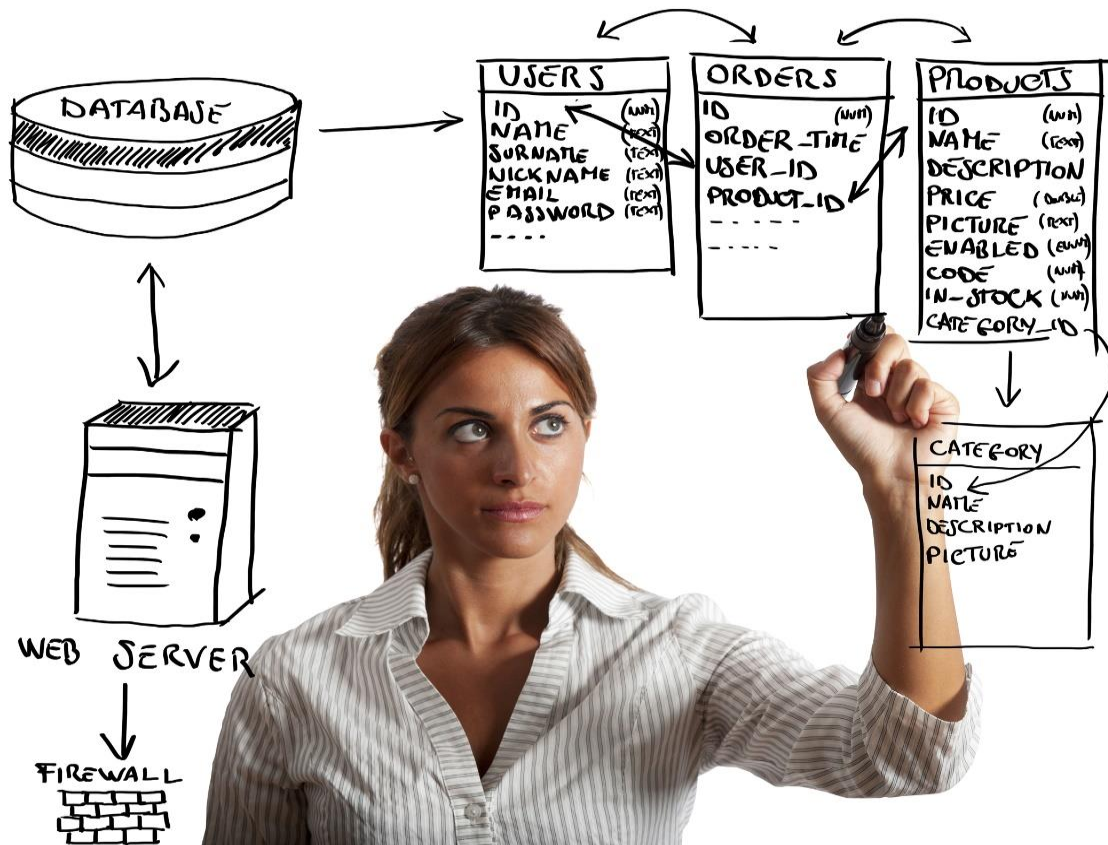


Fig. 1

https://basededatosunounivia.files.wordpress.com/2015/03/shutterstock_116384731.jpg

Una tabla es utilizada para organizar y presentar información. Las tablas se componen de filas y columnas de celdas que se pueden rellenar con textos y gráficos

Las tablas se componen de dos estructuras:

- Registro: es cada una de las filas en que se divide la tabla. Cada registro contiene datos de los mismos tipos que los demás registros. Ejemplo: en una tabla de nombres y direcciones, cada fila contendrá un nombre y una dirección.
- Campo: es cada una de las columnas que forman la tabla. Contienen datos de tipo diferente a los de otros campos. En el ejemplo anterior, un campo contendrá un tipo de datos único, como una dirección, o un número de teléfono, un nombre, etc. A los campos se les puede asignar, además, propiedades especiales que afectan a los registros insertados. El campo puede ser definido como índice o autoincrementable, lo cual permite que los datos de ese campo cambien solos o sea el principal indicador a la hora de ordenar los datos contenidos.

Cada tabla creada debe tener un nombre único en la cada Base de Datos, haciéndola accesible mediante su nombre o su seudónimo (Alias) (dependiendo del tipo de base

de datos elegida) La estructura de las tablas viene dado por la forma de un archivo plano, los cuales en un inicio se componían de un modo similar.

Sintaxis para crear una tabla:

```
CREATE TABLE
    [database_name.[schema_name].| schema_name.]table_name
<column_definition> ::= column_name <data_type>
    [ NULL | NOT NULL ] [ CONSTRAINT constraint_name ] DEFAULT
constant_expression ] | [ IDENTITY [ ( seed ,increment ) ]
<column_constraint> ::=
[ CONSTRAINT constraint_name ]
{   { PRIMARY KEY | UNIQUE }
    [ CLUSTERED | NONCLUSTERED ]
    [ ON { partition_scheme_name ( partition_column_name )
        | filegroup | "default" } ]
    | [ FOREIGN KEY ]
        REFERENCES [schema_name.]referenced_table_name[(column)]
    | CHECK [ NOT FOR REPLICATION ] ( logical_expression )
}
```

Caso Práctico: Crear la tablas Cliente y Producto

```
USE CIBERDATA
GO

/*CREAR EL TIPO DE DATOS*/
CREATE TYPE DATO FROM VARCHAR (100) NOT NULL
GO

CREATE TYPE FECHA FROM DATETIME NOT NUL
GO
```

```
/*CREAR LA TABLA CLIENTE*/
CREATE TABLE CLIENTE
(
    DNICLI CHAR (8) NOT NULL,
    NOMCLI DATO,
    APECLI DATO,
    FECING FECHA,
    FONOCLI VARCHAR (9),
    EMAILCLI VARCHAR (50)
)
GO

/*CREAR LA TABLA PRODUCTO, EL CAMPO IDPROD ES IDENTITY*/
CREATE TABLE PRODUCTO
(
    IDPROD INT IDENTITY (1,100),
    NOMPROD DATO,
    PREUNI DECIMAL,
```

```

    STOCK INT
)
GO

```

1.2.4 Modificación de la estructura de una tabla

Para realizar la modificación de una tabla ejecutamos la sentencia ALTER TABLE, donde:

- Modifica una definición de tabla al alterar, agregar o quitar columnas
- Agregar o quitar y restricciones,
- Reasignar particiones,
- Deshabilitar o habilitar restricciones y desencadenadores.

Sintaxis:

```

ALTER TABLE [database_name.[schema_name]. table_name
{
    ALTER COLUMN column_name
    {
        [ type_schema_name. ] type [ ( { precision [ , scale ]
        [ NULL | NOT NULL ] [ SPARSE ]
    | ADD
    {
        <column_definition>
    | <computed_column_definition>
    | <table_constraint>
    | <column_set_definition>
    } [ ,...n ]

    | DROP
    {
        [ CONSTRAINT ]
        {
            constraint_name
            [ WITH
            ( <drop_clustered_constraint_option> [ ,...n ] )
            ]
        } [ ,...n ]
    | COLUMN
        {
            column_name
        } [ ,...n ]
    } [ ,...n ]
    | [ WITH { CHECK | NOCHECK } ] { CHECK | NOCHECK }
}

```

Caso Práctico: Modificar la table Cliente

```

--AGREGA LAS COLUMNAS A LA TABLA CLIENTE
ALTER TABLE CLIENTE
ADD DIRCLI VARCHAR (255) NOT NULL,
    GICLI VARCHAR (50) NOT NULL

```

GO

--MODIFICANDO LA ESTRUCTURA DE UNA COLUMNA

ALTER TABLE CLIENTE

ALTER COLUMN GICLI VARCHAR (30)

GO

--ELIMINANDO UNA COLUMNA

ALTER TABLE CLIENTE

DROP COLUMN DIRCLI

GO

1.2.5 Manejo de una tabla particionada

Un desafío para un diseñador de base de datos es diseñar sistemas que sean mantenibles, escalables y que tengan un buen desempeño. Una técnica usada para intentar lograr un balance razonable entre mantenimiento y escalabilidad vs. Desempeño es las Tablas Particionadas.

A partir del SQL Server 2008, se provee herramientas para ayudar a los desarrolladores a lograr esta tarea.

Las tablas se deben particionar para dividir grandes cantidades de datos en pequeños bloques para ser manejables y optimizar el rendimiento.

Entre los beneficios que tenemos:

- Más rápido y eficiente acceso a datos.
- El desempeño incrementa para operaciones en paralelo con servidores multiprocesadores, cada procesador puede acceder a múltiples particiones simultáneamente.

Para realizar el proceso de la partición debemos definir una función que determine los límites de los valores los cuales se van a almacenar en la partición:

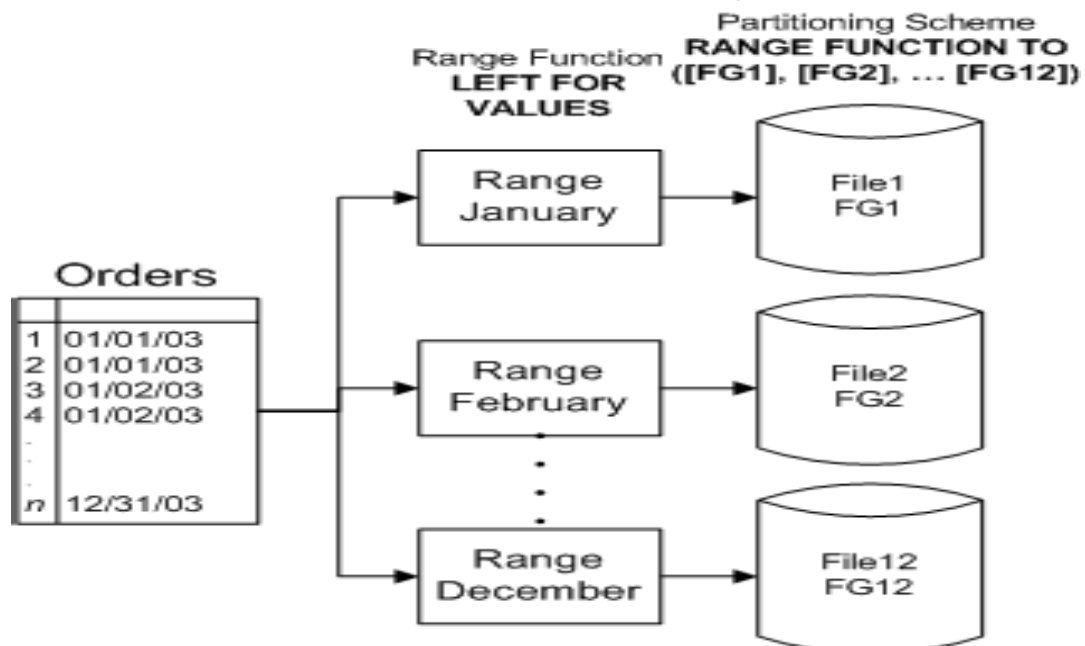


Fig. 2

<https://cubos.files.wordpress.com/2007/07/tablasparticionadas.gif?w=319&h=327>

```

CREATE PARTITION FUNCTION Ejemplo (int)
AS RANGE LEFT FOR VALUES (100, 1000, 10000)
GO

CREATE PARTITION FUNCTION Ejemplo1 (int)
AS RANGE RIGHT FOR VALUES (100, 1000, 10000)
GO

```

Partición	Valores (LEFT)	Valores (RIGHT)
1	<=100	<100
2	>100 AND <=1000	>=100 AND <1000
3	>1000 AND <=10000	>=1000 AND <10000
4	>10000	>=10000

Caso Práctico: Crea una tabla particionada a partir de la función de partición

```

-- Crear una función de partición
CREATE PARTITION FUNCTION PedidosxDecada (datetime)
AS RANGE RIGHT
FOR VALUES ('2000-01-01', '2010-01-01')
GO

-- Filegroup para pedidos < 01/01/2000
ALTER DATABASE BD_Negocios
ADD FILEGROUP Pedidos2000B
GO

-- -- Filegroup para pedidos >= 01/01/2000 AND <=
31/12/2009
ALTER DATABASE BD_Negocios
ADD FILEGROUP Pedidos2010B
GO

-- Filegroup para pedidos >= 01/01/2010
ALTER DATABASE BD_Negocios
ADD FILEGROUP Pedidos2010A
GO

```

A continuación creamos los archivos de base de datos para asociarlos a los FileGroup

```
-- Pedidos2000B filegroup
ALTER DATABASE BD_Negocios
ADD FILE (NAME = Pedidos2000B,
          FILENAME = 'D:\Pedidos2000B.ndf')
TO FILEGROUP Pedidos2000B
GO

-- Pedidos2010B filegroup
ALTER DATABASE BD_Negocios
ADD FILE (NAME = Pedidos2010B,
          FILENAME = 'D:\Pedidos2010B.ndf')
TO FILEGROUP Pedidos2010B
GO

-- Pedidos2010A filegroup
ALTER DATABASE BD_Negocios
ADD FILE (NAME = Pedidos2010A,
          FILENAME = 'D:\Pedidos2010A.ndf')
TO FILEGROUP Pedidos2010A
GO
```

Como siguiente paso defina el schema de la partición la cual será referenciada en la creación de la tabla.

```
/* Crear un partición scheme usando un file group diferente para cada partición*/
CREATE PARTITION SCHEME PedidosxDecadaSC
AS PARTITION PedidosxDecada
TO (Pedidos2000B, Pedidos2010B, Pedidos2010A)
GO

--Crear la tabla donde se utilizara las particiones
CREATE TABLE Pedidos
(
    idPedido int NOT NULL,
    idCliente varchar(20) NULL,
    Fecha datetime NULL,
    Total money NOT NULL)
ON PedidosxDecadaSC(Fecha)
GO

/* Listar los registros de pedidos y visualizar en que particion se encuentra cada
registro*/
SELECT      *
            $PARTITION.PedidosxDecada(FECHA) As [Nro de Particion]
FROM Pedidos
GO
```

1.3. MANEJO DE RESTRICCIONES E ÍNDICES

1.3.1 Constraints o Restricciones. Tipos.

Un constraint es una restricción a un campo de la tabla, sirven para lograr la integridad de datos, aseguramos que los datos ingresados a las columnas sean válidos y que las relaciones entre las tablas no se perderán.

Los constraint pueden definirse al momento de crear la tabla, aunque también es posible hacerlo después de que las tablas se han creado.

Tipos de constraint

NULL | NOT NULL

Determina si se permiten valores NULL en la columna. NULL no es estrictamente una restricción, pero se puede especificar de la misma forma que NOT NULL. NOT NULL se puede especificar para las columnas calculadas solo si se especifica también PERSISTED.

PRIMARY KEY

Es una restricción que exige la integridad de entidad para una o varias columnas especificadas a través de un índice único. Solo se puede crear una restricción PRIMARY KEY para cada tabla.

UNIQUE

Es una restricción que proporciona integridad de entidad para una o varias columnas especificadas a través de un índice único. Las tablas pueden tener múltiples restricciones UNIQUE.

CLUSTERED | NONCLUSTERED

Indica que se ha creado un índice clúster o no agrupado para la restricción PRIMARY KEY o UNIQUE. De forma predeterminada, el valor de las restricciones PRIMARY KEY es CLUSTERED, y el de las restricciones UNIQUE es NONCLUSTERED.

En una instrucción CREATE TABLE, se puede especificar CLUSTERED tan solo para una restricción. Si se especifica CLUSTERED para una restricción UNIQUE y se especifica también una restricción PRIMARY KEY, el valor predeterminado de PRIMARY KEY es NONCLUSTERED.

FOREIGN KEY REFERENCES

Es una restricción que proporciona integridad referencial para los datos de la columna o columnas. Las restricciones FOREIGN KEY requieren que cada valor de la columna exista en la columna o columnas de referencia correspondientes de la tabla a la que se hace referencia. Las restricciones FOREIGN KEY pueden hacer referencia solo a columnas que sean restricciones PRIMARY KEY o UNIQUE en la tabla de referencia o a columnas a las que se haga referencia en UNIQUE INDEX en la tabla de referencia. Las claves externas de las columnas calculadas también se deben marcar como PERSISTED.

CHECK

Es una restricción que exige la integridad del dominio al limitar los valores posibles que se pueden escribir en una o varias columnas. Las restricciones CHECK de las columnas calculadas también se deben marcar como PERSISTED.

DEFAULT

Una restricción DEFAULT permite definir el valor que se proporciona a una columna cuando un usuario no especifica un valor. Por ejemplo, en una tabla con una columna denominada payterms, puede solicitar al servidor de base de datos que especifique "???" o "rellenar más tarde" si el usuario la deja en blanco.

En los diagramas de base de datos, puede definir una restricción DEFAULT como una propiedad de una columna de la tabla. Si desea definir este tipo de restricción para una columna, especifique un valor predeterminado dentro de una tabla en Vista estándar. Asegúrese de especificar la restricción con los delimitadores correctos. Por ejemplo, las cadenas deben estar encerradas entre comillas sencillas.

Sintaxis:

```
ALTER TABLE [database_name.[schema_name]. table_name
{
    ADD | DROP
        [ CONSTRAINT ]
        {
            constraint_name
            [ WITH
                ( <drop_clustered_constraint_option> [ ,...n ] )
            ]
        } [ ,...n ]
    | COLUMN      {
        column_name
    } [ ,...n ]
} [ ,...n ]
| [ WITH { CHECK | NOCHECK } ] { CHECK | NOCHECK }
```

Caso Práctico: Definir las tablas y los constraint para las tablas.

```
--Crear las tablas
Create table categoria
(
    idcat varchar(11) NOT NULL,
    desCat varchar(50) NOT NULL,
    Primary key(idcat)
)
Go

Create Table Proveedor
(
    idprov varchar(10) NOT NULL,
    nomProv varchar(50) NOT NULL,
    rucProv varchar(11) NOT NULL Default('99999999999'),
```

```

        primary key(idprov),
        UNIQUE(nomProv),
        CHECK (rucProv LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
    )
Go

Create Table Producto
(
    idprod varchar(10) NOT NULL,
    nomProd varchar(50) NOT NULL,
    idcat varchar(11) NOT NULL,
    idprov varchar(10) NOT NULL,
    preuni decimal NOT NULL,
    stock int NOT NULL,
    primary key(idprod),
    Foreign Key(idcat) references categoria,
    Foreign Key(idprov) references Proveedor,
    UNIQUE(nomProd),
    CHECK (preUni>=5),
    CHECK (Stock Between 1 and 500)
)
Go

```

Caso Práctico: Creadas las tablas, defina los constraint para cada una de ellas.

```

Create table Cliente
(
    dnicli char(8) not null,
    nomcli dato,
    apecli dato,
    fecing fecha not null,
    fonocli varchar(9),
    emailcli varchar(50)
)
Go

```

```

--Definir los constraint de la tabla
ALTER TABLE Cliente
Add Constraint pkdnicli Primary Key(dnicli)
GO

ALTER TABLE Cliente
Add Constraint chkfecing Check(fecing <=getdate())
GO

ALTER TABLE Cliente
Add Constraint deffecing Default(getdate()) for fecing
GO

ALTER TABLE Cliente

```



```
Add Constraint unk_email UNIQUE(emailcli)
GO
```

```
--Tabla tb_boleta
```

```
Create table tb_boleta(
    num_bol int identity(100,10) not null,
    fec_bol fecha,
    dni_cli char(8) not null,
    monto_bol decimal,
    cancela_bol int
```

```
)
```

```
Go
```

```
--Constraint de Clave primaria
```

```
ALTER TABLE tb_boleta
```

```
Add Constraint pknum_bol Primary Key(num_bol)
```

```
GO
```

```
--fec_bol menor o igual a hoy y por defecto es hoy
```

```
ALTER TABLE tb_boleta
```

```
    Add Default(getdate()) for fec_bol,
```

```
    Check(fec_bol<=getdate())
```

```
Go
```

```
--dni_cli referencia a Cliente
```

```
ALTER TABLE tb_boleta
```

```
Add Constraint fkdni_cli Foreign Key (dni_cli) References Cliente
```

```
Go
```

```
--cancela_bol valores 0 ó 1, por defecto 0
```

```
ALTER TABLE tb_boleta
```

```
Add Constraint defcanc Default(0) for cancela_bol,
```

```
    Constraint chkcanc Check(cancela_bol IN(0,1))
```

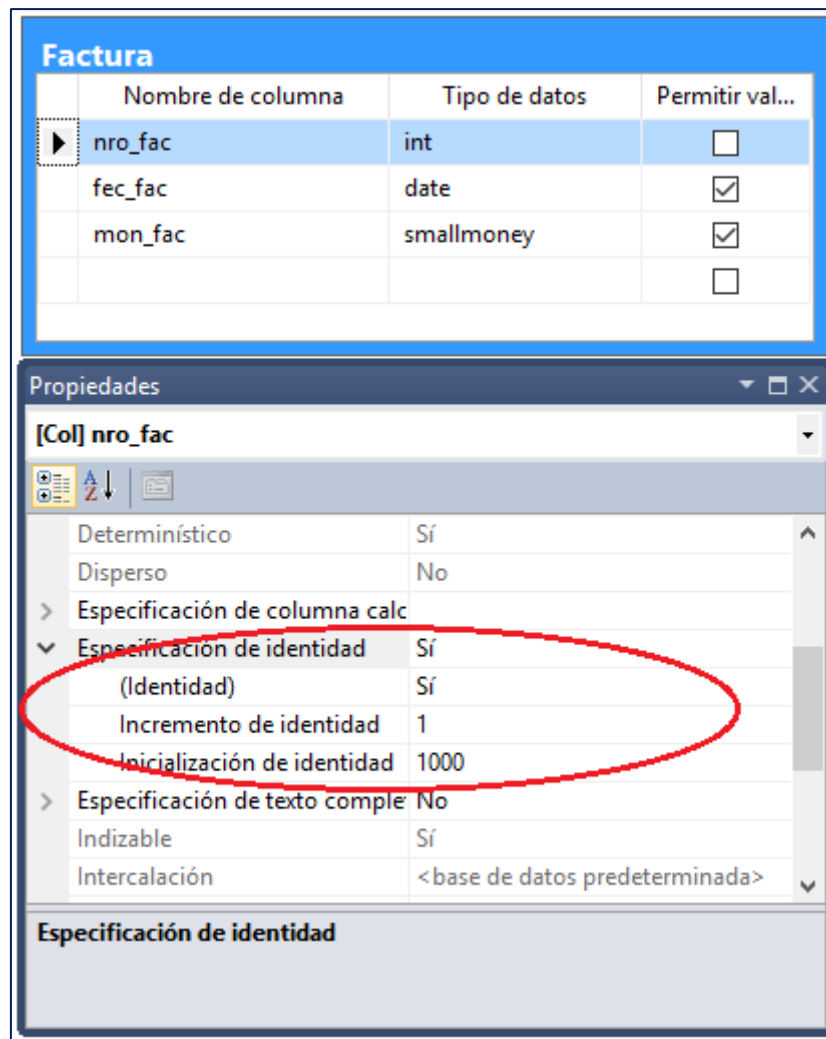
```
Go
```

1.3.2 Uso del Identity

Permite valores explícitos para ser insertado en la columna de identidad de una tabla. Solamente se puede tener un solo identity por cada tabla. Al definir una columna con un identity, definimos el valor incremental al momento de crear su tabla o modificar su estructura.

Adicionalmente, se puede definir una “semilla” que se utilizará como valor inicial, en la primera fila, mientras que se utilizará el valor de “incremento” para ir calculando los siguientes

Para realizar esta tarea desde el Administrador Corporativo, bien en la creación o en la modificación de una tabla, tenemos los campos: identity, inicialización del identity, e incremento del identity.



Podemos utilizar cualquier tipo de dato numérico, en la figura anterior hemos utilizado un int, cuyo valor inicial es 100, y su incremento 1.

En el siguiente ejemplo, crea la misma tabla "alumnos" con un campo que representa un código de identificación que tendrá valores a partir de 100:

```
CREATE TABLE ALUMNOS
(
  IDALU      INT IDENTITY (100,1),
  NOMBRE     VARCHAR (20),
  GRADO      CHAR (5),
  EDAD       TINYINT NULL)
GO
```

En el siguiente ejemplo, se altera una tabla para agregar una columna autoincremental:

```
ALTER TABLE EX_ALUMNOS
  ADD EX_ALUMNO_ID INT IDENTITY (100,1)
GO
```

Para resetear el identity de la tabla, ejecuta la función CheckIdent

```
DBCC CHECKIDENT ('tb_boleta', RESEED, 200)
GO
```

1.3.3 Concepto de Índices- Tipos de Índices.

Un índice es una estructura de datos que permite acceder a diferentes filas de una misma tabla a través de un campo (o campos clave).

Un índice permite un acceso mucho más rápido a los datos.

Para entender cómo trabaja un índice en una tabla; cada tabla se divide en páginas de datos, o bien en páginas a las que podemos acceder rápidamente a través de un índice. Esta idea es la que se aplica en el mundo de las bases de datos, la información está guardada en una tabla (el libro) que tiene muchas hojas de datos (las páginas del libro), con un índice en el que podemos buscar la información que nos interesa.

Si queremos buscar la palabra zapato, ¿qué hacemos?

- Leemos todo el diccionario hasta encontrar la palabra, o
- Buscamos en el índice en que página está la letra z, y es en esa página donde buscamos.

Ni que decir tiene que la opción dos es la correcta, y es de este modo como se utiliza un índice en las bases de datos

Los índices se actualizan automáticamente cuando realizamos operaciones de escritura en la base de datos. Este es un aspecto muy importante de cara al rendimiento de las operaciones de escritura, ya que además de escribir los datos en la tabla se escribirán también en el índice.

Un número elevado de índices hará más lentas estas operaciones. Sin embargo, salvo casos excepcionales, el beneficio que aportan los índices compensa (de largo) esta penalización.

Sintaxis para crear un índice

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]
INDEX index_name ON {tabla | vista} ( column [ ASC | DESC ] [ ,...n
])
[ INCLUDE ( column_name [ ,...n ] ) ]
[ WITH
    [ PAD_INDEX = { ON | OFF }
    [ FILLFACTOR = fillfactor ]
    [ ALLOW_ROW_LOCKS = { ON | OFF } ]
    [ ALLOW_PAGE_LOCKS = { ON | OFF } ]
    [ ON scheme_partition(columna) | FILEGROUP ]
```

Donde

- INCLUDE, Especifica las columnas que no son claves que se añade en el nivel hoja del índice no agrupado.
- PAD_INDEX, especifica el relleno del índice, el valor predeterminado es OFF

- FILLFACTOR, especifica un porcentaje que indica que porcentaje de la hoja debe estar completa, el valor se encuentre entre 1 al 100, por defecto es 0.
- ALLOW_ROW_LOCKS, especifica si los bloqueos de fila se permite, el valor por defecto es ON.
- ALLOW_PAGE_LOCKS, especifica si los bloqueos de página están permitidos, por defecto es ON:

Tipos de Índices

- Índice clúster: ordena y almacena las filas de datos de la tabla o vista por orden en función de la clave del índice clúster. Se implementa como una estructura de árbol B que recupera en forma rápida las filas a partir de los valores de las claves del índice.
- Índice único: garantiza que la clave de índice no contenga valores duplicados y, por tanto, cada fila de la tabla o vista es en cierta forma única.
- Índices no clúster se pueden definir en una tabla o vista con un índice clúster o un montón. Cada fila del índice no clúster contiene un valor de clave no agrupada y un localizador de fila. Este localizador apunta a la fila de datos del índice clúster o el montón que contiene el valor de clave.
Las filas del índice se almacenan en el mismo orden que los valores de la clave del índice, pero no se garantiza que las filas de datos estén en un determinado orden.
- Índices con columnas incluidas es un índice no clúster que se extiende para incluir columnas sin clave además de las columnas claves.

Los índices lo utilizamos cuando:

- Campos autoincrementales (Identity, newsequentialid, etc), deben ser del tipo clustered index. La razón es reducir la (fragmentación).
- Los clustered index son convenientes si se va seleccionar un rango de valores, ordenar (ORDER BY) o agrupar (GROUP BY).
- La clave primaria es un clustered index.
- Para búsquedas de valores específicos, conviene utilizar un nonclustered index.
- Para índices compuestos, mejor non-clustered.

Caso Práctico: Se tiene la tabla tb_alumno, defina los índices siguientes para realizar operaciones de búsqueda.

```
Create Table TB_Alumno
(
    codalu char(5) not null,
    nomalu varchar(50) not null,
    apealu varchar(50) not null,
    fecnac datetime,
    sexalu char(1),
    primary key(codalu)
)
Go

--Índice que realice la búsqueda por apellido y nombre
Create Index Idx_apenom
On TB_alumno (apealu, nomalu)
```

```
Go

--Indice que busca por fecnac, incluya cod, nom y ape
Create Index idx_fecnac
On TB_alumno (fecnac)
INCLUDE (codalu, nomalu, apealu)
Go

--buscar por apealu, donde su fillfactor es 70
Create Index idx_apealu
On TB_alumno (apealu)
With (fillfactor=70)
Go
```

Caso Práctico: Defina la tabla tb_boleta y sus índices.

```
--Crear la tabla tb_boleta
Create Table TB_boleta(
    nbol int identity(1,10) not null,
    fbol datetime,
    cliente varchar(100),
    monto decimal
)
Go

--Defino el constraint de clave primaria
Alter Table TB_boleta
Add Primary Key(nbol)
Go

--Defina el indice por nbol y monto
Create nonclustered index idx_cliente
On TB_boleta(cliente)
Include (nbol, monto)
With (fillfactor=75)
Go
```

```
-- Defina el indice de fecha, para utilizar busquedas
CREATE NONCLUSTERED INDEX IDX_FBOL
ON TB_BOLETA (FBOL)
GO
```

SQL Server proporciona el objeto sys.indexes, el cual permite listar los índices que son administrados por una base de datos.

Aquí tenemos algunos ejemplos:

```
--LISTAR LOS INDICES QUE INICIEN CON IDX
SELECT * FROM SYS.INDEXES
WHERE NAME LIKE 'IDX%'
GO

--LISTAR LOS INDICES DE LA TABLA BOLETA
SELECT T.NAME, I.*
FROM SYS.TABLES T JOIN SYS.INDEXES I
ON T.OBJECT_ID=I.OBJECT_ID
WHERE T.NAME='TB_BOLETA'
GO
```

También podemos eliminar un índice a través del comando DROP INDEX.

```
--Eliminar el índice idx_cliente de tb_boleta
DROP INDEX IDX_CLIENTE
ON TB_BOLETA
GO
```

1.3.4 Índices Particionado.

Aunque los índices con particiones pueden implementarse independientemente de sus tablas, por lo general tiene sentido diseñar una tabla con particiones, luego crear un índice en la tabla.

En consecuencia, en el índice se crean particiones básicamente de la misma forma que en la tabla. De este modo, el índice se alinea con la tabla.

Para ello es necesario crear el Esquema de Particionamiento, el cual, nos permitirá asignar a cada partición el FileGroup que deseamos que utilice para almacenar sus datos. Téngase en cuenta, que al crear el Esquema de Particionamiento, tenemos que asociarlo a una Función de Particionamiento.

Caso Práctico: Crear índice particionado.

```
--Aprovechando el esquema de partición creado anteriormente
--crear un índice particionado por el campo fecha de la tabla Pedidos
CREATE INDEX IDXINDICEPARTICIONADO
ON PEDIDOS (FECHA)
ON PEDIDOSXDECADASC (FECHA)
GO
```

Resumen

- Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso
- Microsoft SQL Server es un sistema para la gestión de bases de datos producido por Microsoft basado en el modelo relacional. Sus lenguajes para consultas son T-SQL y ANSI SQL.
- Para crear una base de datos en SQL Server, se ejecuta la instrucción CREATE DATABASE en modo de confirmación automática (modo predeterminado de administración de transacciones) y no se permite en una transacción explícita o implícita.
- El comando ALTER DATABASE permite modificar los archivos y grupos de archivos asociados con la base de datos. Permite agregar o eliminar los archivos y grupos de archivos de una base de datos, y los cambios de los atributos del archivo de una base de datos.
- Los FileGroups es un concepto muy importante sobre todo cuando vamos a crear una base de datos ya que nos va ayudar mejorar el rendimiento de las consultas y la administración de la data en una BD.

Una tabla es utilizada para organizar y presentar información. Las tablas se componen de filas y columnas de celdas que se pueden rellenar con textos y gráficos; se compone de registros y campos. Cada tabla creada debe tener un nombre único en la cada Base de Datos, haciéndola accesible mediante su nombre o su seudónimo (Alias) (dependiendo del tipo de base de datos elegida) La estructura de las tablas viene dado por la forma de un archivo plano, los cuales en un inicio se componían de un modo similar.

- Un desafío para un diseñador de base de datos es diseñar sistemas que sean mantenibles, escalables y que tengan un buen desempeño.
- Una técnica usada para intentar lograr un balance razonable entre mantenimiento y escalabilidad vs. Desempeño es las Tablas Particionadas.
- Un constraint es una restricción a un campo de la tabla, sirven para lograr la integridad de datos, aseguramos que los datos ingresados a las columnas sean válidos y que las relaciones entre las tablas no se perderán.
- Un identity permite valores explícitos para ser insertado en la columna de identidad de una tabla. Solamente se puede tener un solo identity por cada tabla. Al definir una columna con un identity, definimos el valor incremental al momento de crear su tabla o modificar su estructura.
- Un índice es una estructura de datos que permite acceder a diferentes filas de una misma tabla a través de un campo (o campos clave). Un índice permite un acceso mucho más rápido a los datos. Los índices se actualizan automáticamente cuando realizamos operaciones de escritura en la base de datos.
- Los índices se clasifican en:
 - Clúster
 - UNIQUE
 - NonCluster
 - Índices con columnas incluídas

- SQL Server proporciona el objeto sys.indexes, el cual permite listar los índices que son administrados por una base de datos.
- Aunque los índices con particiones pueden implementarse independientemente de sus tablas, por lo general tiene sentido diseñar una tabla con particiones, luego crear un índice en la tabla.
- Si desea saber más acerca de estos temas, puede consultar las siguientes páginas.
 - <http://msdn.microsoft.com/es-es/library/ms176061.aspx>
Aquí hallará los comandos para crear una base de datos en SQL Server
 - <http://msdn.microsoft.com/en-us/library/bb522469.aspx>
Aquí hallará los comandos para modificar una base de datos en SQL Server
 - <http://msdn.microsoft.com/es-es/library/ms174979.aspx>
Aquí hallará los comandos para crear una tabla en SQL Server



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 utilizando Transact/SQL y aplicando múltiples condiciones de comparación. 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

2.1 Tema 3 : Lenguaje de manipulación de datos DML

- 2.1.1 : Inserción de datos: INSERT y BULK INSERT
- 2.1.2 : Actualización de datos: UPDATE
- 2.1.3 : Eliminación de datos: DELETE
- 2.1.4 : Declaración MERGE

2.2 Tema 4 : Recuperación de Datos

- 2.2.1 : Consulta de datos, uso del SELECT
- 2.2.2 : Ordenar registros
- 2.2.3 : Consultas condicionales, uso de operadores condicionales
- 2.2.4 : Empleo de funciones agregadas: SUM, MIN, MAX, AVG, COUNT.
- 2.2.5 : Uso de las cláusulas GROUP BY y HAVING
- 2.2.5.1 : Opciones CUBE y ROLLUP

2.3 Tema 5 : Recuperación de Datos II

- 2.3.1 : Combinación de tablas
- 2.3.2 : Combinación interna: INNER JOIN
- 2.3.3 : Combinaciones externas: LEFT JOIN, RIGHT JOIN
- 2.3.4 : Combinaciones cruzadas: CROSS JOIN, FULL JOIN
- 2.3.5 : Agregar conjunto de resultados: UNION

ACTIVIDADES PROPUESTAS

1. Los alumnos implementan sentencias SQL para recuperar y actualizar datos en una base de datos relacional.

2. Los alumnos implementan sentencias SQL para agrupar y resumir los datos.

2.1. LENGUAJE DE MANIPULACIÓN DE DATOS

Un lenguaje de Manipulación de Datos (Data Manipulation Language (DML)) es un lenguaje proporcionado por el sistema de gestión de bases de datos que permite a los usuarios de la misma llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado.

El lenguaje de manipulación de datos más popular hoy en día es SQL, usado para recuperar y manipular datos en una base de datos relacional. Otros ejemplos de DML son los usados por bases de datos IMS/DL1, CODASYL u otras.

Se clasifican en dos grandes grupos:

- Lenguajes de consulta procedimentales.
- Lenguajes de consulta no procedimentales

El lenguaje de Consulta Estructurado (Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones sobre las mismas. Una de sus características es el manejo del álgebra y el cálculo relacional permitiendo lanzar consultas con el fin de recuperar información de interés de una base de datos, de una forma sencilla. Es un lenguaje de cuarta generación (4GL)

En este tema vamos a tratar la actualización y recuperación de los datos:

- Insertar nuevas filas,
- Borrar filas o
- Cambiar el contenido de las filas de una tabla.
- Recuperar los registros de las filas de las tablas.

Estas operaciones modifican y consultan los datos almacenados en las tablas pero no su estructura, ni su definición.

2.1.1. Inserción de datos: INSERT y BULK INSERT

INSERT:

Agrega una o varias filas nuevas a una tabla o una vista en SQL Server 2014.

Sintaxis:

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

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

```
SELECT * FROM VENTA.CLIENTES  
GO
```

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 NEGOCIOS  
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 NEGOCIOS  
GO  
  
CREATE TABLE RRHH.EMPLEADOS2011  
(  
    IDEMPLEADO INT NOT NULL,  
    NOMEMPLEADO VARCHAR (50) NOT NULL,  
    APEMPLEADO VARCHAR (50) NOT NULL,  
    FONOEMPLEADO VARCHAR (15) NULL,  
    DIREMPLEADO VARCHAR (100) NOT NULL,  
    IDISTRITO INT NOT NULL  
)  
GO  
  
INSERT INTO RRHH.EMPLEADOS2011  
    SELECT      A.IDEMPLEADO, A.NOMEMPLEADO, A.APEMPLEADO,  
                A.FONOEMPLEADO, A.DIREMPLEADO, A.IDISTRITO  
    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.b.

```
USE NEGOCIOS;
GO

-- CREA UNA VARIABLE TIPO TABLA
DECLARE @PRODUCTO TABLE (
    PRODUCTOID INT NOT NULL,
    PRODUCTONOMBRE VARCHAR (100) NOT NULL,
    PRODUCTOPRE SMALLMONEY,
    PRODUCTOCAN INT);
-- 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 LOS VALORES DE LA VARIABLE TIPO TABLA
SELECT * FROM @PRODUCTO;
GO
```

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

```
USE NEGOCIOS;
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
(COLUMNA_4)
VALUES ('VALOR');

INSERT INTO DBO.PRUEBA
(COLUMNA_2, COLUMNA_4)
VALUES ('VALOR', 'VAL');
```

```

INSERT INTO DBO.PRUEBA
(COLUMNA_2)
VALUES ('VALOR');

INSERT INTO PRUEBA DEFAULT VALUES;
GO

SELECT COLUMNA_1, COLUMNA_2, COLUMNA_3, COLUMNA_4
FROM DBO.PRUEBA;
GO

```

BULK INSERT

Importa un archivo de datos en una tabla o vista de base de datos con un formato especificado por el usuario en SQL Server.

Use esta instrucción para transferir datos eficazmente entre SQL Server y orígenes de datos heterogéneos.

Sintaxis:

```

BULK INSERT
[ [ database_name . [ schema_name ] . | schema_name . ] [ table_name | view_name ]
FROM 'data_file'
[ WITH
(
[[ , ] BATCHSIZE = batch_size ]
[[ , ] CHECK_CONSTRAINTS ]
[[ , ] CODEPAGE = { 'ACP' | 'OEM' | 'RAW' | 'code_page' } ]
[[ , ] DATAFILETYPE =
{ 'char' | 'native' | 'widechar' | 'widenative' } ]
[[ , ] FIELDTERMINATOR = 'field_terminator' ]
[[ , ] FIRSTROW = first_row ]
[[ , ] FIRE_TRIGGERS ]
[[ , ] FORMATFILE = 'format_file_path' ]
[[ , ] KEEPIDENTITY ]
[[ , ] KEEPNULLS ]
[[ , ] KILOBYTES_PER_BATCH = kilobytes_per_batch ]
[[ , ] LASTROW = last_row ]
[[ , ] MAXERRORS = max_errors ]
[[ , ] ORDER ( { column [ ASC | DESC ] } [ ,...n ] ) ]
[[ , ] ROWS_PER_BATCH = rows_per_batch ]
[[ , ] ROWTERMINATOR = 'row_terminator' ]
[[ , ] TABLOCK ]
[[ , ] ERRORFILE = 'file_name' ]
)]

```

Caso Práctico: Insertar datos desde un archivo plano.

```
BULK INSERT negocios.clientes
FROM 'D:\clientes.txt'
WITH (FIELDTERMINATOR=';');
```

```
BULK INSERT Venta.Pedido
FROM 'F:\orders\PEDIDO.txt'
WITH
(
    FIELDTERMINATOR = '|',
    ROWTERMINATOR = '\n'
);
```

Este ejemplo, inserta registros desde un archivo plano y ejecuta los triggers de la table destino.

```
BULK INSERT Venta.Pedido
FROM 'f:\orders\Ordentxt'
WITH
(
    FIELDTERMINATOR = '|',
    ROWTERMINATOR = '\n',
    FIRE_TRIGGERS
);
```

2.1.2. 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.

Sintaxis:

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

Donde:

- 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.

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

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

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 NEGOCIOS;  
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
```

2.1.3. 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}]
```

Donde:

- 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.

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 NEGOCIOS;  
GO  
  
DELETE FROM VENTA.PEDIDOSCABE  
GO
```

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

2.1.4. Sentencia 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:

- 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.
- 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 run>
```

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 NEGOCIOS  
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 NEGOCIOS
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 NEGOCIOS
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 VALUES (SOURCE.NOMBRECLIENTE, SOURCE .DIRCLIENTE)
WHEN NOT MATCHED BY SOURCE THEN
    DELETE;
GO

```

2.2. RECUPERACIÓN DE DATOS

2.2.1. Consulta de datos. Uso del 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.

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:

```

TH 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
```

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

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 NEGOCIOS  
GO
```

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

GO

SELECT * FROM DBO.EMPLEADODAK
GO
```

2.2.2. Ordenar registros

Se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula:

ORDER BY (Lista de Campos) [ASC|DESC]

Por ejemplo: Listar los registros de pedidos ordenados por cliente en forma ascendente

```
SELECT *
FROM PEDIDOSCABE
ORDER BY IDCLIENTE ASC
GO
```

Por ejemplo: Listar los registros de pedidos ordenados por cliente en forma ascendente y por fecha en forma descendente.

```
SELECT *
FROM TB_PEDIDOSCABE
ORDER BY IDCLIENTE ASC, FECHAPEDIDO DESC
GO
```

Listar los 10 productos más caros

```
SELECT TOP 10 *
FROM TB_PRODUCTOS
ORDER BY PRECIOUNIDAD DESC
GO
```

2.2.3. Consultas condicionales, uso de operadores condicionales

En las operaciones anteriores se vio la forma de recuperar los registros de las tablas, las formas empleadas devolvían todos los registros de la mencionada tabla.

En esta sesión se estudiarán las posibilidades de filtrar los registros con el fin de recuperar solamente aquellos que cumplan unas condiciones preestablecidas.

La cláusula condicional

- Determina qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT.
- En esta cláusula se deben especificar las condiciones expuestas en los puntos anteriores de la presente sesión.
- Si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla.

Las condiciones de búsqueda o calificaciones de las cláusulas WHERE pueden incluir los siguientes operadores:

- Operadores de Comparación
- Operadores Lógicos
- Operadores: Intervalos de valores
- Operadores: Listas de valores
- Operadores: Coincidencias de patrón

Operadores de Comparación

- Comprueban la veracidad de alguna condición.
- Devuelven el tipo de datos Boolean con el valor TRUE o FALSE.
- Los operadores de comparación soportados por SQL son: >, >=, <, <=, =, <>

Operadores lógicos

- Comprueban la veracidad de alguna condición.
- Devuelven el tipo de datos Boolean con el valor TRUE, FALSE o UNKNOWN.
- Los operadores lógicos soportados por SQL son: AND, OR, NOT

Tienen la siguiente sintaxis:

<i>Expresion1 OPERADOR Expresion2</i>

Donde:

Expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico.

Operadores de intervalos de valores

Recuperan los registros según el intervalo de valores de un campo

Empleamos el operador Between cuya sintaxis es:

<i>Campo [Not] Between Valor1 AND Valor2</i>
--

Operadores de lista de valores

Permite seleccionar las filas que coinciden con alguno de los valores de una lista

Los operadores de lista soportados por SQL son: In, NOT In, cuya sintaxis es la siguiente:

Campo [Not] In (valor1, valor2,... valor n)

Operadores de coincidencia de patrones

Permite buscar valores de cadenas de caracteres que coincidan con un patrón determinado

El operador de coincidencias es LIKE, la cual está compuesta por 4 caracteres comodín

Comodín	Significado
%	Cualquier cadena de cero más caracteres
-	Cualquier carácter
[]	Cualquier carácter individual del intervalo, por ejemplo [a-f]
[^]	Cualquier carácter individual fuera del intervalo, por ejemplo [^a-f]

Ejemplos prácticos

--Listar los clientes cuyo nombre tenga a la letra P en su expresión

```
SELECT *
FROM TB_CLIENTES
WHERE NOMBRECIA LIKE '%P%'
GO
```

--Listar los pedidoscabe cuyo año de la fechapedido se encuentre entre 2004 hasta -- el presente año

```
SELECT *
FROM TB_PEDIDOSCABE
WHERE YEAR (FECHAPEDIDO) BETWEEN 2004 AND YEAR (GETDATE ())
GO
```

--Listar todos aquellos proveedores que tienen productos

```
SELECT *
FROM TB_PROVEEDORES V
WHERE NOT EXISTS (SELECT * FROM TB_PRODUCTOS P
                  WHERE P.IDPROVEEDOR=V.IDPROVEEDOR)
GO
```

--Listar los datos de los clientes, de aquellos clientes

--que residen en los países de Chile, USA o Perú

```
SELECT *
FROM TB_CLIENTES
WHERE IDPAIS IN (SELECT P.IDPAIS
                FROM TB_PAISES P
                WHERE P.NOMBREPAIS IN ('PERÚ','CHILE','USA'))
GO
```



```
--Listar los países donde no hemos registrado clientes
SELECT *
FROM TB_PAISES P
WHERE NOT EXISTS (SELECT * FROM TB_CLIENTES C
                  WHERE P.IDPAIS=C.IDPAIS)
GO
```

```
--Listar los pedidoscabe registrados en la mitad de este
--año y cuyos nombre de clientes inicien con A
SELECT *
FROM TB_PEDIDOSCABE
WHERE      MONTH (FECHAPEDIDO) <=6 AND
          YEAR (FECHAPEDIDO) = YEAR (GETDATE ()) AND
          IDCLIENTE IN (SELECT C.IDCLIENTE
                        FROM TB_CLIENTES C
                        WHERE NOMBRECIA LIKE 'A%')
GO
```

```
--Listar los clientes que no han realizado pedido este año
SELECT *
FROM TB_CLIENTES C
WHERE NOT EXISTS (SELECT IDCLIENTE
                  FROM TB_PEDIDOSCABE P
                  WHERE P.IDCLIENTE=C.IDCLIENTE AND
                        YEAR (P.FECHAPEDIDO) =YEAR (GETDATE ()))
GO
```

```
--Listar todos los productos que fueron comprados en este año (uso de IN)
SELECT DISTINCT P.*
FROM TB_PRODUCTOS P, TB_PEDIDOSCABE PC, TB_PEDIDOSDETA PD
WHERE      P.IDPRODUCTO = PD.IDPRODUCTO AND
          PC.IDPEDIDO = PD.IDPEDIDO AND
          YEAR (PC.FECHAPEDIDO)=2011
GO
--otro método----
SELECT DISTINCT P.*
FROM TB_PRODUCTOS P
WHERE EXISTS (SELECT *
             FROM TB_PEDIDOSDETA D JOIN TB_PEDIDOSCABE C
             ON C.IDPEDIDO=D.IDPEDIDO
             WHERE YEAR (C.FECHAPEDIDO) = 2011 AND
                   D.IDPRODUCTO=P.IDPRODUCTO)
GO
```

2.2.4. Empleo de funciones agregadas: SUM, MIN, MAX, AVG, COUNT

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.

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 NEGOCIOS
GO

SELECT COUNT (*) AS 'CANTIDAD DE PEDIDOS'
FROM 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 NEGOCIOS
GO
```

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

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 calculada 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 NEGOCIOS
GO
```

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

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 NEGOCIOS
GO
```

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

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 NEGOCIOS
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
```

2.2.5. Uso de las cláusulas GROUP BY y HAVING.

Cláusula GROUP BY

Agrupar 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.

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 NEGOCIOS
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 cláusula 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 NEGOCIOS

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
```

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 cláusula 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 NEGOCIOS

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 NEGOCIOS
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

```

2.2.5.1. Opciones CUBE y ROLLUP.

Las consultas que usan los operadores ROLLUP y CUBE generan algunos de los conjuntos de resultados y realizan algunos de los cálculos que lleva a cabo las aplicaciones OLAP.

El operador CUBE genera un conjunto de resultados que se puede utilizar en los informes de tabulación cruzada.

Una operación ROLLUP puede calcular el equivalente de una dimensión o jerarquía OLAP.

Por ejemplo, dada una dimensión de tiempo con los niveles o atributos año, mes y día, la siguiente operación ROLLUP genera las agrupaciones siguientes.

Operación	Agrupaciones
ROLLUP (DATEPART(yyyy,OrderDate) ,DATEPART(mm,OrderDate) ,DATEPART(dd,OrderDate))	year, month, day year, month year ()

Dada una dimensión de ubicación con los niveles región y ciudad concatenados con los niveles de dimensión de tiempo año, mes y día, la siguiente operación ROLLUP genera las agrupaciones siguientes.

Operación	Agrupaciones
ROLLUP (region, city), ROLLUP (DATEPART(yyyy,OrderDate) ,DATEPART(mm,OrderDate) ,DATEPART(dd,OrderDate))	region, city, year, month, day region, city, year, month region, city, year region, city region, year, month, day region, year, month region, year region year, month, day

	year, month year ()
--	----------------------------

Una operación CUBE de los mismos niveles desde las dimensiones de ubicación y tiempo da como resultado las agrupaciones siguientes.

Operación	Agrupaciones
CUBE (region, city ,DATEPART(yyyy,OrderDate) ,DATEPART(mm,OrderDate) ,DATEPART(dd,OrderDate))	region, city, year, month, day region, city, year, month region, city, year region, city region, city, month, day region, city, month region, city, day region, city, year, day region, city, day region, year, month, day region, year, month region, year region, month, day region, month region, year, day region, day region city, year, month, day city, year, month city, year city, month, day city, month city, year, day city, day year, month, day year, month year year, day month, day month day ()

Los ejemplos utilizan la función de agregado SUM para que se puedan comparar los conjuntos de resultados. También se podrían utilizar las otras funciones de agregado para calcular resúmenes diferentes.

Caso Práctico: Usar un GROUP BY simple

En el ejemplo siguiente, el GROUP BY simple devuelve un conjunto de resultados para comparar con los conjuntos de resultados de los ejemplos siguientes. Estos ejemplos utilizan los operadores GROUP BY con la misma instrucción SELECT.

USE Negocios
GO

```
SELECT    PA.NombrePais,
          PV.NomProveedor,
          CA.NombreCategoria,
          PR.NomProducto,
          SUM (PE.Cantidad) [CANTIDAD PEDIDA]
FROM      Ventas.países PA JOIN Compras.proveedores PV
          ON PA.Idpais = PV.idpais JOIN Compras.productos PR
          ON PV.IdProveedor = PR.IdProveedor JOIN Compras.categorias CA
          ON CA.IdCategoria = PR.IdCategoria JOIN Ventas.pedidosdeta PE
          ON PR.IdProducto = PE.IdProducto
WHERE     PA.NombrePais IN ('Colombia','Peru')
GROUP BY  PA.NombrePais,
          PV.NomProveedor,
          CA.NombreCategoria,
          PR.NomProducto
ORDER BY  PA.NombrePais,
          PV.NomProveedor,
          CA.NombreCategoria,
          PR.NomProducto
GO
```

Este sería el resultado:

NombrePais	NomProveedor	NombreCategoria	NomProducto	CANTIDAD PEDIDA
Colombia	Cooperativa de Quesos Las Cabras	Lacteos	Queso Cabrales	706
Colombia	Cooperativa de Quesos Las Cabras	Lacteos	Queso Manchego La Pastora	344
Colombia	Gai pâturage	Lacteos	Camembert Pierrot	1577
Colombia	Gai pâturage	Lacteos	Raclet de queso Courdavault	1496
Colombia	Plutzer Lebensmittelgroßmärkte AG	Bebidas	Cerveza Klosterbier Rhönbräu	1155
Colombia	Plutzer Lebensmittelgroßmärkte AG	Carnes	Salchicha Thüringer	746
Colombia	Plutzer Lebensmittelgroßmärkte AG	Condimentos	Salsa verde original Frankfurter	791
Colombia	Plutzer Lebensmittelgroßmärkte AG	Frutas/Verduras	Col fermentada Rössle	640
Colombia	Plutzer Lebensmittelgroßmärkte AG	Granos/Cereales	Bollos de pan de Wimmer	740
Colombia	Tokyo Traders	Carnes	Buey Mishi Kobe	95
Colombia	Tokyo Traders	Frutas/Verduras	Queso de soja Longlife	297
Colombia	Tokyo Traders	Pescado/Marisco	Pez espada	742
Peru	Grandma Kellys Homestead	Condimentos	Mermelada de grosellas de la abuela	301
Peru	Grandma Kellys Homestead	Condimentos	Salsa de arandanos Northwoods	372
Peru	Grandma Kellys Homestead	Frutas/Verduras	Peras secas organicas del tio Bob	763
Peru	Karkki Oy	Bebidas	Licor Cloudberry	981
Peru	Karkki Oy	Reposteria	Chocolate blanco	235
Peru	Karkki Oy	Reposteria	Regaliz	520
Peru	Leka Trading	Bebidas	Cafe de Malasia	580
Peru	Leka Trading	Condimentos	Azúcar negra Malacca	601
Peru	Leka Trading	Granos/Cereales	Tallarines de Singapur	697

Caso Práctico: Usar un GROUP BY ROLLUP


```
USE Negocios
GO

SELECT      PA.NombrePais,
            PV.NomProveedor,
            CA.NombreCategoria,
            PR.NomProducto,
            SUM (PE.Cantidad) [CANTIDAD PEDIDA]
FROM Ventas.países PA JOIN Compras.proveedores PV
    ON PA.Idpais = PV.idpais JOIN Compras.productos PR
    ON PV.IdProveedor = PR.IdProveedor JOIN Compras.categorias CA
    ON CA.IdCategoria = PR.IdCategoria JOIN Ventas.pedidosdeta PE
    ON PR.IdProducto = PE.IdProducto
WHERE PA.NombrePais IN ('Colombia','Peru')
GROUP BY ROLLUP (PA.NombrePais,
                PV.NomProveedor,
                CA.NombreCategoria,
                PR.NomProducto)
ORDER BY PA.NombrePais,
        PV.NomProveedor,
        CA.NombreCategoria,
        PR.NomProducto
GO
```

Este sería el resultado devuelve las siguientes agrupaciones:

- NombrePais, NomProveedor, NombreCategoria, NomProducto
- NombrePais, NomProveedor, NombreCategoria
- NombrePais, NomProveedor
- NombrePais
- Total general

NombrePais	NomProveedor	NombreCategoria	NomProducto	CANTIDAD PEDIDA
NULL	NULL	NULL	NULL	14379
Colombia	NULL	NULL	NULL	9329
Colombia	Cooperativa de Quesos Las Cabras	NULL	NULL	1050
Colombia	Cooperativa de Quesos Las Cabras	Lacteos	NULL	1050
Colombia	Cooperativa de Quesos Las Cabras	Lacteos	Queso Cabrales	706
Colombia	Cooperativa de Quesos Las Cabras	Lacteos	Queso Manchego La Pastora	344
Colombia	Gai pâturage	NULL	NULL	3073
Colombia	Gai pâturage	Lacteos	NULL	3073
Colombia	Gai pâturage	Lacteos	Camembert Pierrot	1577
Colombia	Gai pâturage	Lacteos	Raclet de queso Courdavault	1496
Colombia	Plutzer Lebensmittelgroßmärkte AG	NULL	NULL	4072
Colombia	Plutzer Lebensmittelgroßmärkte AG	Bebidas	NULL	1155
Colombia	Plutzer Lebensmittelgroßmärkte AG	Bebidas	Cerveza Klosterbier Rhönbräu	1155
Colombia	Plutzer Lebensmittelgroßmärkte AG	Carnes	NULL	746
Colombia	Plutzer Lebensmittelgroßmärkte AG	Carnes	Salchicha Thüringer	746
Colombia	Plutzer Lebensmittelgroßmärkte AG	Condimentos	NULL	791
Colombia	Plutzer Lebensmittelgroßmärkte AG	Condimentos	Salsa verde original Frankfurter	791
Colombia	Plutzer Lebensmittelgroßmärkte AG	Frutas/Verduras	NULL	640
Colombia	Plutzer Lebensmittelgroßmärkte AG	Frutas/Verduras	Col fermentada Rössle	640
Colombia	Plutzer Lebensmittelgroßmärkte AG	Granos/Cereales	NULL	740
Colombia	Plutzer Lebensmittelgroßmärkte AG	Granos/Cereales	Bollos de pan de Wimmer	740
Colombia	Tokyo Traders	NULL	NULL	1134
Colombia	Tokyo Traders	Carnes	NULL	95
Colombia	Tokyo Traders	Carnes	Buey Mishi Kobe	95
Colombia	Tokyo Traders	Frutas/Verduras	NULL	297
Colombia	Tokyo Traders	Frutas/Verduras	Queso de soja Longlife	297
Colombia	Tokyo Traders	Pescado/Marisco	NULL	742
Colombia	Tokyo Traders	Pescado/Marisco	Pez espada	742
Peru	NULL	NULL	NULL	5050
Peru	Grandma Kellys Homestead	NULL	NULL	1436
Peru	Grandma Kellys Homestead	Condimentos	NULL	673
Peru	Grandma Kellys Homestead	Condimentos	Mermelada de grosellas de la abuela	301
Peru	Grandma Kellys Homestead	Condimentos	Salsa de arandanos Northwoods	372
Peru	Grandma Kellys Homestead	Frutas/Verduras	NULL	763
Peru	Grandma Kellys Homestead	Frutas/Verduras	Peras secas organicas del tio Bob	763
Peru	Karkki Oy	NULL	NULL	1736
Peru	Karkki Oy	Bebidas	NULL	981
Peru	Karkki Oy	Bebidas	Licor Cloudberry	981
Peru	Karkki Oy	Reposteria	NULL	755
Peru	Karkki Oy	Reposteria	Chocolate blanco	235
Peru	Karkki Oy	Reposteria	Regaliz	520
Peru	Leka Trading	NULL	NULL	1878
Peru	Leka Trading	Bebidas	NULL	580
Peru	Leka Trading	Bebidas	Cafe de Malasia	580
Peru	Leka Trading	Condimentos	NULL	601
Peru	Leka Trading	Condimentos	Azúcar negra Malacca	601
Peru	Leka Trading	Granos/Cereales	NULL	697
Peru	Leka Trading	Granos/Cereales	Tallarines de Singapur	697

Caso Práctico: Usar un GROUP BY CUBE

USE Negocios
GO

```

SELECT    PA.NombrePais,
          PV.NomProveedor,
          CA.NombreCategoria,
          PR.NomProducto,
          SUM (PE.Cantidad) [CANTIDAD PEDIDA]
FROM Ventas.países PA JOIN Compras.proveedores PV
      ON PA.Idpais = PV.idpais JOIN Compras.productos PR
      ON PV.IdProveedor = PR.IdProveedor JOIN Compras.categorias CA
      ON CA.IdCategoria = PR.IdCategoria JOIN Ventas.pedidosdeta PE
      ON PR.IdProducto = PE.IdProducto
WHERE PA.NombrePais IN ('Colombia','Peru')
GROUP BY CUBE (PA.NombrePais,
              PV.NomProveedor,
              CA.NombreCategoria,
              PR.NomProducto)
ORDER BY PA.NombrePais,
         PV.NomProveedor,
         CA.NombreCategoria,
         PR.NomProducto
GO

```

El resultado sería:

NombrePais	NomProveedor	NombreCategoria	NomProducto	CANTIDAD PEDIDA
NULL	NULL	NULL	NULL	14379
NULL	NULL	NULL	Azúcar negra Malacca	601
NULL	NULL	NULL	Bollos de pan de Wimmer	740
NULL	NULL	NULL	Buey Mishi Kobe	95
NULL	NULL	NULL	Cafe de Malasia	580
NULL	NULL	NULL	Camembert Pierrot	1577
NULL	NULL	NULL	Cerveza Klosterbier Rhönbräu	1155
NULL	NULL	NULL	Chocolate blanco	235
NULL	NULL	NULL	Col fermentada Rössle	640
NULL	NULL	NULL	Licor Cloudberry	981
NULL	NULL	NULL	Mermelada de grosellas de la abuela	301
NULL	NULL	NULL	Peras secas organicas del tio Bob	763
NULL	NULL	NULL	Pez espada	742
NULL	NULL	NULL	Queso Cabrales	706
NULL	NULL	NULL	Queso de soja Longlife	297
NULL	NULL	NULL	Queso Manchego La Pastora	344
NULL	NULL	NULL	Raclet de queso Courdavault	1496
NULL	NULL	NULL	Regaliz	520
NULL	NULL	NULL	Salchicha Thüringer	746
NULL	NULL	NULL	Salsa de arandanos Northwoods	372
NULL	NULL	NULL	Salsa verde original Frankfurter	791
NULL	NULL	NULL	Tallarines de Singapur	697
NULL	NULL	Bebidas	NULL	2716
NULL	NULL	Bebidas	Cafe de Malasia	580
NULL	NULL	Bebidas	Cerveza Klosterbier Rhönbräu	1155
NULL	NULL	Bebidas	Licor Cloudberry	981
NULL	NULL	Carnes	NULL	841
NULL	NULL	Carnes	Buey Mishi Kobe	95
NULL	NULL	Carnes	Salchicha Thüringer	746

NULL	NULL	Condimentos	NULL	2065
NULL	NULL	Condimentos	Azúcar negra Malacca	601
NULL	NULL	Condimentos	Mermelada de grosellas de la abuela	301
NULL	NULL	Condimentos	Salsa de arandanos Northwoods	372
NULL	NULL	Condimentos	Salsa verde original Frankfurter	791
NULL	NULL	Frutas/Verduras	NULL	1700
NULL	NULL	Frutas/Verduras	Col fermentada Rössle	640
NULL	NULL	Frutas/Verduras	Peras secas organicas del tio Bob	763
NULL	NULL	Frutas/Verduras	Queso de soja Longlife	297
NULL	NULL	Granos/Cereales	NULL	1437
NULL	NULL	Granos/Cereales	Bollos de pan de Wimmer	740
NULL	NULL	Granos/Cereales	Tallarines de Singapur	697
NULL	NULL	Lacteos	NULL	4123
NULL	NULL	Lacteos	Camembert Pierrot	1577
NULL	NULL	Lacteos	Queso Cabrales	706
NULL	NULL	Lacteos	Queso Manchego La Pastora	344
NULL	NULL	Lacteos	Raclet de queso Courdavault	1496
NULL	NULL	Pescado/Marisco	NULL	742
NULL	NULL	Pescado/Marisco	Pez espada	742
NULL	NULL	Reposteria	NULL	755
NULL	NULL	Reposteria	Chocolate blanco	235
NULL	NULL	Reposteria	Regaliz	520
NULL	Cooperativa de Quesos Las Cabras	NULL	NULL	1050
NULL	Cooperativa de Quesos Las Cabras	NULL	Queso Cabrales	706
NULL	Cooperativa de Quesos Las Cabras	NULL	Queso Manchego La Pastora	344
NULL	Cooperativa de Quesos Las Cabras	Lacteos	NULL	1050
NULL	Cooperativa de Quesos Las Cabras	Lacteos	Queso Cabrales	706
NULL	Cooperativa de Quesos Las Cabras	Lacteos	Queso Manchego La Pastora	344
NULL	Gai pâturage	NULL	NULL	3073
NULL	Gai pâturage	NULL	Camembert Pierrot	1577
NULL	Gai pâturage	NULL	Raclet de queso Courdavault	1496
NULL	Gai pâturage	Lacteos	NULL	3073
NULL	Gai pâturage	Lacteos	Camembert Pierrot	1577
NULL	Gai pâturage	Lacteos	Raclet de queso Courdavault	1496
NULL	Grandma Kellys Homestead	NULL	NULL	1436
NULL	Grandma Kellys Homestead	NULL	Mermelada de grosellas de la abuela	301
NULL	Grandma Kellys Homestead	NULL	Peras secas organicas del tio Bob	763
NULL	Grandma Kellys Homestead	NULL	Salsa de arandanos Northwoods	372
NULL	Grandma Kellys Homestead	Condimentos	NULL	673
NULL	Grandma Kellys Homestead	Condimentos	Mermelada de grosellas de la abuela	301
NULL	Grandma Kellys Homestead	Condimentos	Salsa de arandanos Northwoods	372
NULL	Grandma Kellys Homestead	Frutas/Verduras	NULL	763
NULL	Grandma Kellys Homestead	Frutas/Verduras	Peras secas organicas del tio Bob	763
NULL	Karkki Oy	NULL	NULL	1736
NULL	Karkki Oy	NULL	Chocolate blanco	235
NULL	Karkki Oy	NULL	Licor Cloudberry	981
NULL	Karkki Oy	NULL	Regaliz	520
NULL	Karkki Oy	Bebidas	NULL	981
NULL	Karkki Oy	Bebidas	Licor Cloudberry	981
NULL	Karkki Oy	Reposteria	NULL	755
NULL	Karkki Oy	Reposteria	Chocolate blanco	235
NULL	Karkki Oy	Reposteria	Regaliz	520
NULL	Leka Trading	NULL	NULL	1878
NULL	Leka Trading	NULL	Azúcar negra Malacca	601
NULL	Leka Trading	NULL	Cafe de Malasia	580
NULL	Leka Trading	NULL	Tallarines de Singapur	697
NULL	Leka Trading	Bebidas	NULL	580
NULL	Leka Trading	Bebidas	Cafe de Malasia	580
NULL	Leka Trading	Condimentos	NULL	601
NULL	Leka Trading	Condimentos	Azúcar negra Malacca	601
NULL	Leka Trading	Granos/Cereales	NULL	697
NULL	Leka Trading	Granos/Cereales	Tallarines de Singapur	697
NULL	Plutzer Lebensmittelgroßmärkte AG	NULL	NULL	4072

2.3. RECUPERACIÓN DE DATOS II

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.

2.3.1. Combinación 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 puede especificar en las cláusulas FROM o WHERE, aunque se recomienda que se especifiquen en la cláusula FROM.

Las combinaciones se pueden clasificar en:

- Combinación Interna
- Combinación Externa

2.3.2. Combinación interna: INNER JOIN

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 NEGOCIOS
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 NEGOCIOS
GO

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

2.3.3. Combinación externa.

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 utiliza las siguientes palabras claves 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

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 NEGOCIOS
GO

SELECT C.*, P.IDPEDIDO
FROM VENTAS.CLIENTES C LEFT JOIN VENTAS.PEDIDOS CABE P
ON C.IDCLIENTE = P.IDCLIENTE
GO
```

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 NEGOCIOS
GO

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

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 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 aún no se ha registrado en algún pedido.

```
USE NEGOCIOS
GO
```

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

2.3.4. Combinación 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 NEGOCIOS
GO
```

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

2.3.5. 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.

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

```
USE NEGOCIOS  
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 NEGOCIOS
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
```

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.
- 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.
- JOIN, en el lenguaje de consulta, permite combinar registros de dos o más tablas en una base de datos relacional. JOIN se puede 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.
- 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.



INTRODUCCIÓN A LA PROGRAMACIÓN TRANSACT SQL

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno construye programas estructurados y maneja los errores utilizando el lenguaje Transact-SQL.

TEMARIO

3.1 Tema 7 : Sentencias SQL para la programación

- 3.1.1 : Fundamentos de la programación con Transact - SQL
- 3.1.2 : Identificadores
- 3.1.3 : Variables: declaración, asignación
- 3.1.4 : Elementos de flujo de control
 - 3.1.4.1 : Estructura de control IF
 - 3.1.4.2 : Estructura de control CASE
 - 3.1.4.3 : Estructura de control WHILE
- 3.1.5 : Control de errores con TRY / CATCH, uso de @@Error, uso del RaisError
- 3.1.6 : Uso de transacción: Commit y RollBack

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.

3.1. SENTENCIAS SQL PARA LA PROGRAMACIÓN

3.1.1. Fundamentos de la programación con Transact SQL

SQL es un lenguaje de consulta para los sistemas de bases de datos relacionales, 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.

3.1.2. Identificadores

En Transact SQL los identificadores de variables deben comenzar por el carácter @, es decir, el nombre de una variable debe comenzar por @. Para declarar variables en Transact SQL debemos utilizar la palabra clave declare, seguido del identificador y tipo de datos de la variable.

Los identificadores deben tener una longitud entre 1 y 128 caracteres, no deben de contener espacios en blanco

Excepciones:

Ciertos objetos, como las tablas temporales, tienen identificadores de menor tamaño. Para versiones anteriores a MS SQL Server 7.0, los identificadores eran de 30 caracteres como máximo.

No se pueden usar palabras reservadas para identificadores:

Add, all, alter, and, any, as, asc, authorization, backup, begin, between, break, browse, bulk, by, cascade, case, check, checkpoint, close, clustered, coalesce, collate, column, commit, compute, constraint, contains, containstable, continue, convert, create, cross, current, current_date, current_time, current_timestamp, current_user, cursor, database, dbcc, deallocate, declare, default, delete, deny, desc, disk, distinct, distributed, double, drop, dummy, dump, else, end, errlvl, escape, except, exec, execute, exists, exit, fetch, file, fillfactor, for, foreign, freetext, freetexttable, from, full, function, goto, grant, group, having, holdlock, identity, identity_insert, identitycol, if, in, index, inner, Insert, intersect, into, is, join, key, kill, left, like, lineno, load, national, nocheck, nonclustered, not, null, nullif, of, off, offsets, on, open, opendatasource, openquery, openrowset, openxml, option, or, order, outer, over, percent, plan, precision, primary, print, proc, procedure, public, raiserror, read, readtext, reconfigure, references, replication, restore, restrict, return, revoke, right, rollback, rowcount, rowguidcol, rule, save, schema, select, session_user, set, setuser, shutdown, some, statistics, system_user, table, textsize, then, to, top, tran, ,transaction, trigger, truncate,

tsequal, union, unique, update, updatetext, use, user, values, varying, view, ,waitfor, when, where, while, with, writetext.

3.1.3. Variables: declaración, asignación.

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.

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.

```
USE Negocios  
GO  
  
DECLARE @PRECIO DECIMAL  
SET @PRECIO = 50  
SELECT * FROM Compras.PRODUCTOS P  
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 NEGOCIOS  
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

```

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 de 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 <i>insert</i>
@@VERSION	Devuelve la Versión del <i>SQL Server</i>
@@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

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

```

3.1.4. Elementos de flujo de control.

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

3.1.4.1. Estructura 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 código) ha realizado pedidos.

```

USE NEGOCIOS
GO

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 Ventas.países WHERE IDPAIS = @COPAIS)
BEGIN
    UPDATE Ventas.países
    SET NOMBREPAIS = @NOMBRE
    WHERE IDPAIS = @COPAIS
END
ELSE
BEGIN
INSERT INTO ventas.países VALUES (@COPAIS, @NOMBRE)
END
GO
```

3.1.4.2. 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 número 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 'SEPTIEMBRE'
    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 título de cortesía asignando, para cada valor, una expresión.

```
Use Northwind
GO

Select (Case TitleOfCourtesy
        When 'Ms.' Then 'Señorita'
        When 'Mrs.' Then 'Señora'
        When 'Mr.' Then 'Señor'
        When 'Dr.' Then 'Doctor'
        Else ''
      End) as [Tratamiento],
      LastName as [Apellido],
      FirstName as [Nombre]
From Employees
GO
```

El resultado sería:

	Tratamiento	Apellido	Nombre
1	Señorita	Davolio	Nancy
2	Doctor	Fuller	Andrew
3	Señorita	Leverling	Janet
4	Señora	Peacock	Margaret
5	Señor	Buchanan	Steven
6	Señor	Suyama	Michael
7	Señor	King	Robert
8	Señorita	Callahan	Laura
9	Señorita	Dodsworth	Anne

Usar una instrucción SELECT con 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.

```

USE NEGOCIOS
GO

DECLARE @STOCK INT
SET @STOCK=100

SELECT NOMPRODUCTO, PRECIOUNIDAD, UNIDADES EN EXISTENCIA,
       'ESTADO' = (CASE
                   WHEN UNIDADES EN EXISTENCIA > @STOCK THEN 'STOCKEADO'
                   WHEN UNIDADES EN EXISTENCIA = @STOCK THEN 'LIMITE'
                   WHEN UNIDADES EN EXISTENCIA < @STOCK THEN 'HAGA UNA SOLICITUD'
                   END)
FROM COMPRAS.PRODUCTOS
GO

```

El resultado sería:

Resultados		Mensajes		
	NomProducto	PRECIO UNIDAD	UNIDADES EN EXISTENCIA	ESTADO
1	Te Dharamsala	18	39	HAGA UNA SOLICITUD
2	Cerveza tibetana Barley	19	17	HAGA UNA SOLICITUD
3	Sirope de regaliz	10	13	HAGA UNA SOLICITUD
4	Espicias Cajun del chef Anton	22	53	HAGA UNA SOLICITUD
5	Mezcla Gumbo del chef Anton	21	0	HAGA UNA SOLICITUD
6	Mermelada de grosellas de la abuela	25	120	STOCKEADO
7	Peras secas organicas del tio Bob	30	15	HAGA UNA SOLICITUD
8	Salsa de arandanos Northwoods	40	6	HAGA UNA SOLICITUD
9	Buey Mishi Kobe	97	29	HAGA UNA SOLICITUD
10	Pez espada	31	31	HAGA UNA SOLICITUD
11	Queso Cabrales	21	22	HAGA UNA SOLICITUD

✓ Consulta ejecutada correctamente.

3.1.4.3 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.

```
USE NEGOCIOS
GO

DECLARE @COUNTER INT = 1;
WHILE @COUNTER < 6
BEGIN
    SELECT TOP (1) P.IDPRODUCTO, P.NOMPRODUCTO, P.PRECIOUNIDAD
    FROM COMPRAS.PRODUCTOS AS 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
```

Ejemplo: Mientras el precio promedio del producto es menor de 300, se duplica los precios. Luego, si el precio máximo es mayor a 500, sale del bucle, case contrario, continua el bucle.

```
USE NORTHWIND
GO

WHILE (SELECT AVG (UNITPRICE) FROM PRODUCTS) < 300
    BEGIN
        --ACTUALIZA
        UPDATE PRODUCTS
        SET UNITPRICE = UNITPRICE * 2
        --MUESTRA EL PRECIO MAS ALTO
        SELECT MAX (UNITPRICE) FROM PRODUCTS
        --EVALUA EL PRECIO MAS ALTO
        IF (SELECT MAX (UNITPRICE) FROM PRODUCTS) > 500
            BREAK
        ELSE
            CONTINUE
    END
GO
```

3.1.5. Control de errores con TRY / CATCH, uso de @@Error, uso de RaisError.

TRY / CATCH

Implementa un mecanismo de control de errores para Transact-SQL que es similar al control de excepciones en los lenguajes Microsoft Visual C# y Microsoft Visual C++. Se puede incluir un grupo de instrucciones Transact-SQL en un bloque TRY. Si se produce un error en el bloque TRY, el control se transfiere a otro grupo de instrucciones que está incluido en un bloque CATCH.

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 = 10
    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;

```

Errores no afectados por una construcción TRY...CATCH

Las construcciones TRY...CATCH no detectan lo siguiente:

- Advertencias o mensajes informativos que tienen una gravedad 10 o inferior.
- Errores que tienen la gravedad 20 o superior que detienen el procesamiento de las tareas de Motor de base de datos de SQL Server en la sesión. Si se produce un error con una gravedad 20 o superior y no se interrumpe la conexión con la base de datos, TRY...CATCH controlará el error.
- Atenciones, como solicitudes de interrupción de clientes o conexiones de cliente interrumpidas.
- Cuando el administrador del sistema finaliza la sesión mediante la instrucción KILL.

Un bloque CATCH no controla los siguientes tipos de errores cuando se producen en el mismo nivel de ejecución que la construcción TRY...CATCH:

- Errores de compilación, como errores de sintaxis, que impiden la ejecución de un lote.
- Errores que se producen durante la recopilación de instrucciones, como errores de resolución de nombres de objeto que se producen después de la compilación debido a una resolución de nombres diferida.

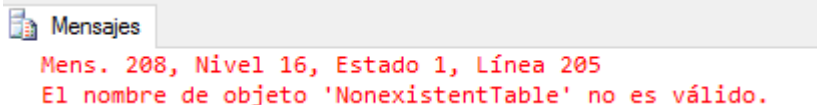
En el ejemplo siguiente se muestra cómo la construcción TRY...CATCH no captura un error de resolución de nombre de objeto generado por una instrucción SELECT.

```

BEGIN TRY
    -- Tabla no existe
    -- error no capturado y es enviado al nivel superior.
    SELECT * FROM NonexistentTable;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS NroError,
        ERROR_MESSAGE() AS MensajeError;
END CATCH

```

Muestra un mensaje capturado por el nivel superior.



Mensajes

Mens. 208, Nivel 16, Estado 1, Línea 205
El nombre de objeto 'NonexistentTable' no es válido.

En este ejemplo, al ejecutar la instrucción SELECT dentro de un procedimiento almacenado, el error se produce en un nivel inferior al bloque TRY. La construcción TRY...CATCH controlará el error.

```
-- Verifica la existencia del procedimiento.
IF OBJECT_ID ('usp_ExampleProc') IS NOT NULL
    DROP PROCEDURE usp_ExampleProc;
GO
```

```
-- Crea un procedimiento que provoca un error
-- de nombre de objeto.
CREATE PROCEDURE usp_ExampleProc
AS
    SELECT * FROM NonexistentTable;
GO

BEGIN TRY
    EXECUTE usp_ExampleProc;
END TRY
BEGIN CATCH
    SELECT
        ERROR_NUMBER() AS NroError,
        ERROR_MESSAGE() AS MensajeError;
END CATCH;
```

Muestra el mensaje capturado por Try / Catch, debido a que se encuentra en un nivel superior al del procedimiento almacenado.

Resultados		Mensajes
NroError	MensajeError	
1	208	El nombre de objeto 'NonexistentTable' no es válido.

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 número 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
-----------------	------------------------------

Uso de @@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

```
USE NEGOCIOS
GO

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

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

```
Use Negocios
go

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

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.

```
USE NEGOCIOS
GO

BEGIN TRY
    DELETE FROM compras.PRODUCTOS
    WHERE UNIDADESENEXISTENCIA < 20
END TRY
BEGIN CATCH
    PRINT 'NO SE PUEDE ELIMINAR ESTE PRODUCTO'
END CATCH
GO
```

```
END TRY
BEGIN CATCH
    DECLARE @MENSAJE VARCHAR (255)
    --RECUPERAR LA DESCRIPCION DEL VALOR DE @@ERROR
    SELECT @MENSAJE= M.DESCRPTION
    FROM SYS.SYMESSAGES M WHERE M.ERROR=@@ERROR
    PRINT @MENSAJE
END CATCH
```

Uso de 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.

Sintaxis:

```
RAISERROR ( { msg_id | msg_str | @local_variable }
    { ,severity ,state }
    [ ,argument [ ,...n ] ] )
    [ WITH option [ ,...n ] ]
```

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
GO
```

3.1.6. Uso de transacciones: Commit y RollBack.

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.

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 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 el siguiente ejemplo se va a realizar una transacción explícita.
Primero consultamos la tabla Países

```
USE NEGOCIOS
GO

SELECT * FROM VENTAS.PAISES
GO
```

Tenemos este resultado. Revise el registro seleccionado.

Resultados		Mensajes
	Idpais	NombrePais
1	001	Peru
2	002	Argentina
3	003	Chile
4	004	USA
5	005	España
6	006	Francia
7	007	Colombia
8	008	Canada
9	009	China
10	99	ESPAÑA

Luego ejecutamos la transacción explícita

```

USE NEGOCIOS
GO

DECLARE @V_IDPAIS CHAR (3) = '99'
DECLARE @V_NOMPAIS VARCHAR (50) = 'UGANDA'
BEGIN TRAN MITRANSACCION
    UPDATE VENTAS.PAISES
    SET NOMBREPAIS = @V_NOMPAIS
    WHERE IDPAIS = @V_IDPAIS
    IF @V_IDPAIS > 80
        ROLLBACK TRAN MITRANSACCION
    ELSE
        COMMIT TRAN MITRANSACCION
GO

```

Vemos que la transacción (Update) no ha tenido efecto, debido a que la condición revierte el cambio (RollBack).

Resultados		Mensajes
	Idpais	NombrePais
1	001	Peru
2	002	Argentina
3	003	Chile
4	004	USA
5	005	España
6	006	Francia
7	007	Colombia
8	008	Canada
9	009	China
10	99	ESPAÑA

Ahora ejecutamos esta transacción

```

USE NEGOCIOS
GO

DECLARE @V_IDPAIS CHAR (3) = '99'
DECLARE @V_NOMPAIS VARCHAR (50) = 'UGANDA'
BEGIN TRAN MITRANSACCION
    UPDATE VENTAS.PAISES
    SET NOMBREPAIS = @V_NOMPAIS
    WHERE IDPAIS = @V_IDPAIS
    IF @V_IDPAIS > 100
        ROLLBACK TRAN MITRANSACCION
    ELSE
        COMMIT TRAN MITRANSACCION
GO

```

Vemos que la transacción (Update) si ha tenido efecto, debido a que la condición confirma la transacción (Commit).

Resultados		Mensajes
	Idpais	NombrePais
1	001	Peru
2	002	Argentina
3	003	Chile
4	004	USA
5	005	España
6	006	Francia
7	007	Colombia
8	008	Canada
9	009	China
10	99	Uganda

Resumen

- SQL es un lenguaje de consulta para los sistemas de bases de datos relacionales, 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.
- 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.devjoker.com/gru/Tutorial-Transact-SQL/TSQL/Tutorial-TransactSQL.aspx>
Aquí hallará los conceptos de programación TRANSACT SQL.



CREACIÓN DE CURSORES

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno construye programas avanzados utilizando el lenguaje Transact/SQL que incorporen cursores con la capacidad de procesar grandes volúmenes de transacciones con el mejor rendimiento posible.

TEMARIO

4.1 Tema 8 : Construcción de Cursores

- 4.1.1 : Construcción de cursores explícitos e implícitos
- 4.1.2 : Cursores y actualización de datos

ACTIVIDADES PROPUESTAS

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

4.1 CONSTRUCCIÓN DE CURSORES

En bases de datos, el término cursor se refiere a una estructura de control utilizada para el recorrido (y potencial procesamiento) de los registros obtenidos del resultado de una consulta.

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

Las operaciones de una base de datos relacional actúan en un conjunto completo de filas. Por ejemplo, el conjunto de filas que devuelve una instrucción SELECT está compuesto por todas las filas que satisfacen las condiciones de la cláusula WHERE de la instrucción. Este conjunto completo de filas que devuelve la instrucción se conoce como conjunto de resultados. Las aplicaciones, especialmente las aplicaciones interactivas en línea, no siempre trabajan de forma eficaz con el conjunto de resultados completo si lo toman como una unidad. Estas aplicaciones necesitan un mecanismo que trabaje con una fila o un pequeño bloque de filas cada vez. Los cursores son una extensión de los conjuntos de resultados que proporcionan dicho mecanismo.

Los cursores amplían el procesamiento de los resultados porque:

- Permiten situarse en filas específicas del conjunto de resultados.
- Recuperan una fila o un bloque de filas de la posición actual en el conjunto de resultados.
- Aceptan modificaciones de los datos de las filas en la posición actual del conjunto de resultados.
- Aceptan diferentes grados de visibilidad para los cambios que realizan otros usuarios en la información de la base de datos que se presenta en el conjunto de resultados.
- Proporcionan instrucciones Transact-SQL en scripts, procedimientos almacenados y acceso de desencadenadores a los datos de un conjunto de resultados.

La siguiente información puede variar dependiendo de los sistemas gestores de bases de datos.

Recuperar una fila del cursor puede resultar en un retraso, conocido en inglés como network round trip, y que se debe al tiempo necesario para enviar la petición al SGBD y esperar los datos. Ello supone la utilización de mucho más ancho de banda de la red de lo que se necesitaría por norma general para ejecutar una sola sentencia SQL como DELETE. Repetidos network round trips pueden suponer un impacto severo en la velocidad de operación del cursor. Algunos SGBD intentan minimizar este impacto utilizando el fetch de bloque. Un fetch de bloque implica que se envían múltiples filas o registros de forma conjunta desde el servidor al cliente. El cliente almacena el bloque de fila en un buffer local y recupera las filas desde el mismo hasta que el buffer está vacío.

Los cursores reservan recursos en el servidor, como por ejemplo locks, packages, procesos, almacenamiento temporal, etc. Por ejemplo, Microsoft SQL Server implementa los cursores creando una tabla temporal y rellenándola con los datos de la consulta. Si un cursor no se cierra de manera correcta, el recurso no será liberado hasta que la sesión SQL (conexión) sea cerrada. Este desperdicio de recursos en el servidor puede llevar no sólo a una degradación del rendimiento, sino también a fallos más graves.

4.1.1 Construcción de cursores explícitos e implícitos

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 implementados por instrucciones TRANSACTSQL (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>**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 **DELETE** 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.

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>

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.

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.

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

```
USE NEGOCIOS
GO

--DECLARA EL CURSOR
DECLARE MI_CURSOR CURSOR FOR
    SELECT * FROM COMPRAS.PRODUCTOS

--ABRIR
OPEN MI_CURSOR

--IMPRIMIR EL PRIMER REGISTRO
FETCH NEXT FROM MI_CURSOR

--CERRAR EL CURSOR
CLOSE MI_CURSOR

--LIBERAR EL CURSOR
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.

```
-- DECLARANDO CURSOR
```

```
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 CURSOR
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.

```
USE NEGOCIOS
GO

-- 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.NomCliente, 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).

```
USE NEGOCIOS  
GO  
  
-- 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.NomCliente, COUNT (*)  
    FROM VENTAS.CLIENTES C JOIN VENTAS.PEDIDOSCABE PC  
        ON C.IDCLIENTE=PC.IDCLIENTE  
    GROUP BY C.NomCliente  
  
-- 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+  
            SPACE (40-LEN (@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 REPLICATE ('=',45)
PRINT 'TOTAL DE PEDIDOS:' + STR (@TOTAL)
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 = 0
-- DECLARO EL CURSOR
DECLARE MI_CURSOR CURSOR FOR
        SELECT YEAR (C.FECHAPEDIDO),
                C.IDPEDIDO, SUM (D.PRECIOUNIDAD*D.CANTIDAD)
        FROM VENTAS.PEDIDOSCABE C JOIN VENTAS.PEDIDOSDETA D
        ON C.IDPEDIDO=D.IDPEDIDO
        GROUP BY YEAR (FECHAPEDIDO), C.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)
PRINT REPLICATE ('*',20)

-- 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 REPLICATE ('-',30)
                PRINT 'IMPORTE EN ' + CAST (@Y1 AS VARCHAR) +
                    SPACE (2) + ': ES ' + CAST (@TOTAL AS VARCHAR)
                PRINT SPACE (10)
                PRINT 'AÑO:' + CAST (@Y AS VARCHAR)
                PRINT REPLICATE ('*', 20)
                SET @Y1=@Y
            END
    END
```

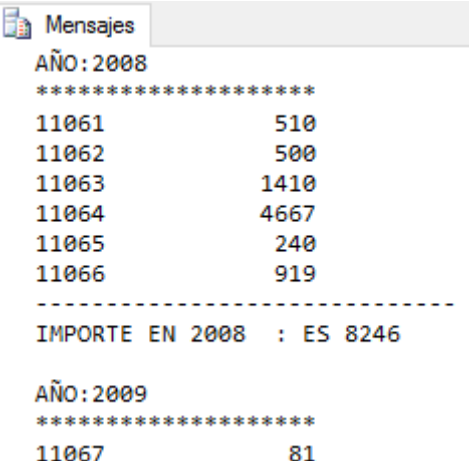
```

        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 EL ÚLTIMO TOTAL
PRINT REPLICATE('-',30)
PRINT 'IMPORTE EN '+CAST (@Y1 AS VARCHAR)+SPACE (2)+
      ': ES '+STR (@TOTAL)

```

El resultado sería así:



```

Mensajes
AÑO: 2008
*****
11061          510
11062          500
11063         1410
11064         4667
11065          240
11066          919
-----
IMPORTE EN 2008 : ES 8246

AÑO: 2009
*****
11067          81

```

4.1.2 Cursores de actualización de datos.

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, NOMPRODUCTO, PRECIOUNIDAD,
           UNIDADES EN EXISTENCIA
    FROM COMPRAS.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 COMPRAS.PRODUCTOS
    SET PRECIOUNIDAD = @PRECIO
    WHERE CURRENT OF CPRODUCTO
    --IMPRIMIR
    PRINT 'EL PRECIO DE ' + @NOMBRE + ' ES ' + LTRIM(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
```

El resultado sería:

 Mensajes

```
(1 filas afectadas)
EL PRECIO DE Te Dharamsala ES 7

(1 filas afectadas)
EL PRECIO DE Cerveza tibetana Barley ES 8

(1 filas afectadas)
EL PRECIO DE Sirope de regaliz ES 4

(1 filas afectadas)
EL PRECIO DE Especias Cajun del chef Anton ES 9
```

Resumen

- Un cursor es una variable que nos permite recorrer 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



PROGRAMACIÓN TRANSACT SQL

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno implementa instrucciones Transact/SQL y de programación mediante procedimientos almacenados, funciones para optimizar las operaciones en la base de datos y desencadenadores para optimizar las operaciones (insert, delete y update) de registros en una base de datos.

TEMARIO

5.1 Tema 9 : Manejo de Procedimientos Almacenados

- 5.1.1 : Definición y tipos
- 5.1.2 : Construcción de procedimientos almacenados
- 5.1.3 : Manejo de parámetros: valores de entrada, valores de retorno

5.2 Tema 10 : Manejo de Procedimientos Almacenados II

- 5.2.1 : Anidamiento de procedimientos almacenados
- 5.2.2 : Operaciones con procedimientos almacenados

5.3 Tema 11 : Manejo de Funciones de Usuario

- 5.3.1 : Funciones del sistema
- 5.3.2 : Funciones de usuario
 - 5.3.2.1 : Funciones escalares
 - 5.3.2.2 : Funciones de tabla
 - 5.3.2.3 : Funciones multisentencias

5.4 Tema 12 : Desencadenadores

5.4.1 : Desencadenadores DML

5.4.2 : Desencadenadores DDL

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.

5.1 MANEJO DE PROCEDIMIENTOS ALMACENADOS

5.1.1 Definición y tipos

Un procedimiento almacenado de SQL Server es un grupo de una o varias instrucciones Transact-SQL o una referencia a un método de Common Runtime Language (CLR) de Microsoft .NET Framework. Los procedimientos se asemejan a las construcciones de otros lenguajes de programación, porque pueden:

- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida al programa que realiza la llamada.
- Contener instrucciones de programación que realicen operaciones en la base de datos. Entre otras, pueden contener llamadas a otros procedimientos.
- Devolver un valor de estado a un programa que realiza una llamada para indicar si la operación se ha realizado correctamente o se han producido errores, y el motivo de estos.

Ventajas:

Algunas de las ventajas que brinda el uso de procedimientos son.

- Tráfico de red reducido entre el cliente y el servidor
- Los comandos de un procedimiento se ejecutan en un único lote de código. Esto puede reducir significativamente el tráfico de red entre el servidor y el cliente porque únicamente se envía a través de la red la llamada que va a ejecutar el procedimiento. Sin la encapsulación de código que proporciona un procedimiento, cada una de las líneas de código tendría que enviarse a través de la red.
- Mayor seguridad
- Varios usuarios y programas cliente pueden realizar operaciones en los objetos de base de datos subyacentes a través de un procedimiento, aunque los usuarios y los programas no tengan permisos directos sobre esos objetos subyacentes. El procedimiento controla qué procesos y actividades se llevan a cabo y protege los objetos de base de datos subyacentes. Esto elimina la necesidad de conceder permisos en cada nivel de objetos y simplifica los niveles de seguridad.
- La cláusula EXECUTE AS puede especificarse en la instrucción CREATE PROCEDURE para habilitar la suplantación de otro usuario o para permitir que los usuarios o las aplicaciones puedan realizar ciertas actividades en la base de datos sin necesidad de contar con permisos directos sobre los objetos y comandos subyacentes. Por ejemplo, algunas acciones como TRUNCATE TABLE no tienen

permisos que se puedan conceder. Para poder ejecutar TRUNCATE TABLE, el usuario debe tener permisos ALTER en la tabla especificada. Puede que la concesión de permisos ALTER a un usuario en una tabla no sea lo ideal, pues en realidad el usuario tendrá permisos muy superiores a la posibilidad de truncar una tabla. Si se incorpora la instrucción TRUNCATE TABLE en un módulo y se especifica la ejecución del módulo como un usuario con permisos para modificar la tabla, se pueden ampliar los permisos para truncar la tabla al usuario al que se concedan permisos EXECUTE para el módulo.

- Al llamar a un procedimiento a través de la red, solo está visible la llamada que va a ejecutar el procedimiento. Por lo tanto, los usuarios malintencionados no pueden ver los nombres de los objetos de base de datos y tabla, incrustados en sus propias instrucciones Transact-SQL, ni buscar datos críticos.
- El uso de parámetros de procedimientos ayuda a protegerse contra ataques por inyección de código SQL. Dado que la entrada de parámetros se trata como un valor literal y no como código ejecutable, resulta más difícil para un atacante insertar un comando en la instrucción Transact-SQL del procedimiento y comprometer la seguridad.
- Los procedimientos pueden cifrarse, lo que ayuda a ofuscar el código fuente. Para obtener más información, vea Cifrado de SQL Server.
- Reutilización del código
- El código de cualquier operación de base de datos redundante resulta un candidato perfecto para la encapsulación de procedimientos. De este modo, se elimina la necesidad de escribir de nuevo el mismo código, se reducen las inconsistencias de código y se permite que cualquier usuario o aplicación que cuente con los permisos necesarios pueda acceder al código y ejecutarlo.
- Mantenimiento más sencillo
- Cuando las aplicaciones cliente llaman a procedimientos y mantienen las operaciones de base de datos en la capa de datos, solo deben actualizarse los cambios de los procesos en la base de datos subyacente. El nivel de aplicación permanece independiente y no tiene que tener conocimiento sobre los cambios realizados en los diseños, las relaciones o los procesos de la base de datos.
- Rendimiento mejorado
- De forma predeterminada, un procedimiento se compila la primera vez que se ejecuta y crea un plan de ejecución que vuelve a usarse en posteriores ejecuciones. Como el procesador de consultas no tiene que crear un nuevo plan, normalmente necesita menos tiempo para procesar el procedimiento.
- Si ha habido cambios importantes en las tablas o datos a los que se hace referencia en el procedimiento, el plan precompilado podría hacer que el procedimiento se ejecutara con mayor lentitud. En este caso, volver a crear el procedimiento y forzar un nuevo plan de ejecución puede mejorar el rendimiento.

Tipos de Procedimientos.-

Definidos por el usuario

Un procedimiento definido por el usuario se puede crear en una base de datos definida por el usuario o en todas las bases de datos del sistema excepto en la base de datos Resource. El procedimiento se puede desarrollar en Transact-SQL o como una

referencia a un método de Common Runtime Language (CLR) de Microsoft .NET Framework.

Temporales

Los procedimientos temporales son una forma de procedimientos definidos por el usuario. Los procedimientos temporales son iguales que los procedimientos permanentes salvo porque se almacenan en tempdb. Hay dos tipos de procedimientos temporales: locales y globales. Se diferencian entre sí por los nombres, la visibilidad y la disponibilidad. Los procedimientos temporales locales tienen como primer carácter de sus nombres un solo signo de número (#); solo son visibles en la conexión actual del usuario y se eliminan cuando se cierra la conexión. Los procedimientos temporales globales presentan dos signos de número (##) antes del nombre; son visibles para cualquier usuario después de su creación y se eliminan al final de la última sesión en la que se usa el procedimiento.

Sistema

Los procedimientos del sistema se incluyen con SQL Server. Están almacenados físicamente en la base de datos interna y oculta Resource y se muestran de forma lógica en el esquema sys de cada base de datos definida por el sistema y por el usuario. Además, la base de datos msdb también contiene procedimientos almacenados del sistema en el esquema dbo que se usan para programar alertas y trabajos. Dado que los procedimientos del sistema empiezan con el prefijo sp_, le recomendamos que no use este prefijo cuando asigne un nombre a los procedimientos definidos por el usuario. Para obtener una lista completa de los procedimientos del sistema, vea Procedimientos almacenados del sistema (Transact-SQL).

SQL Server admite los procedimientos del sistema que proporcionan una interfaz de SQL Server a los programas externos para varias actividades de mantenimiento. Estos procedimientos extendidos usan el prefijo xp_. Para obtener una lista completa de los procedimientos extendidos, vea Procedimientos almacenados extendidos generales (Transact-SQL).

Extendidos definidos por el usuario

Los procedimientos extendidos permiten crear rutinas externas en un lenguaje de programación como C. Estos procedimientos son archivos DLL que una instancia de SQL Server puede cargar y ejecutar dinámicamente.

5.1.2 Construcción de procedimientos almacenados

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

```
USE NEGOCIOS  
GO  
  
-- CREA PROCEDIMIENTO ALMACENADO  
CREATE PROCEDURE USP_CLIENTES  
AS  
SELECT IDCLIENTE AS CODIGO,  
        NOMCLIENTE AS CLIENTE,  
        DIRCLIENTE AS DIRECCION,  
        FONOCLIENTE AS 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:

```
EXECUTE USP_CLIENTES  
GO
```

O

```
EXEC USP_CLIENTES  
GO
```

O simplemente:

```
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 01 parámetro de entrada y para ejecutarlo deberá usar la siguiente sentencia:

```
-- EJECUTANDO EL PROCEDIMIENTO ALMACENADO
EXEC USP_PEDIDOSFECHAS @F1='08-07-1996'
GO
```

El resultado será:

	IdPedido	IdCliente	IdEmpleado	FechaPedido	FechaEntrega	FechaEnvio	EnvioP
1	10250	HANAR	4	1996-07-08 00:00:00.000	1996-08-05 00:00:00.000	1996-07-12 00:00:00.000	1
2	10251	VICTE	3	1996-07-08 00:00:00.000	1996-08-05 00:00:00.000	1996-07-15 00:00:00.000	1

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_PEDIDOSFECHAS '08-07-1996'
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_PEDIDOSFECHAS
@F1 DATETIME,
@F2 DATETIME
AS
    SELECT *
    FROM VENTAS.PEDIDOSCABE
    WHERE FECHAPEDIDO BETWEEN @F1 AND @F2
GO
```

Para ejecutar el procedimiento almacenado:

```
EXEC USP_PEDIDOSFECHAS @F1='08-07-1996', @F2='15-07-1996'
GO
```

El resultado sería:

Resultados		Mensajes					
	IdPedido	IdCliente	IdEmpleado	FechaPedido	FechaEntrega	FechaEnvio	EnvioP
1	10250	HANAR	4	1996-07-08 00:00:00.000	1996-08-05 00:00:00.000	1996-07-12 00:00:00.000	1
2	10251	VICTE	3	1996-07-08 00:00:00.000	1996-08-05 00:00:00.000	1996-07-15 00:00:00.000	1
3	10252	SUPRD	4	1996-07-09 00:00:00.000	1996-08-06 00:00:00.000	1996-07-11 00:00:00.000	1
4	10253	HANAR	3	1996-07-10 00:00:00.000	1996-07-24 00:00:00.000	1996-07-16 00:00:00.000	1
5	10254	CHOPS	5	1996-07-11 00:00:00.000	1996-08-08 00:00:00.000	1996-07-23 00:00:00.000	1
6	10255	RICSU	9	1996-07-12 00:00:00.000	1996-08-09 00:00:00.000	1996-07-15 00:00:00.000	1
7	10256	WELLI	3	1996-07-15 00:00:00.000	1996-08-12 00:00:00.000	1996-07-17 00:00:00.000	1

Para eliminar un procedimiento almacenado, ejecute la instrucción DROP PROCEDURE

```
DROP PROCEDURE USP_PEDIDOSFECHAS
GO
```

5.1.3 Manejo de parámetros: valores de entrada, valores de retorno.

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.

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 TRANSACTSQL, 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.

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.

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: 2011.

```
USE NEGOCIOS
GO

CREATE PROCEDURE USP_PEDIDOSCLIENTEAÑO
@ID VARCHAR (5),
@AÑO INT = 2011
AS
BEGIN
    SELECT      PC.IDPEDIDO AS 'PEDIDO',
               FECHAPEDIDO, NOMPRODUCTO,
               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
END
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
```

El resultado será:

	PEDIDO	FECHAPEDIDO	NOMPRODUCTO	PRECIO	CANTIDAD
1	10643	1997-08-25 00:00:00.000	Col fermentada Rössle	45	15
2	10643	1997-08-25 00:00:00.000	Licor verde Chartreuse	18	21
3	10643	1997-08-25 00:00:00.000	Arenque salado	12	2
4	10692	1997-10-03 00:00:00.000	Sandwich de vegetales	43	20
5	10702	1997-10-13 00:00:00.000	Sirope de regaliz	10	6
6	10702	1997-10-13 00:00:00.000	Licor Cloudberry	18	15

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.

```
USE NEGOCIOS
```

```
GO
```

```
CREATE PROCEDURE USP_REPORTEPEDIDOSEMPLEADO
```

```
@ID INT,
```

```
@Y INT,
```

```
@Q INT OUTPUT,
```

```
@MONTO DECIMAL OUTPUT
```

```
AS
```

```
BEGIN
```

```
    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
```

```
END
```

```
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 @v_Q INT, @v_M DECIMAL
```

```
EXEC USP_REPORTEPEDIDOSEMPLEADO
```

```
    @ID=2,
```

```
    @Y=1997,
```

```
    @Q=@v_Q      OUTPUT,
```

```
    @MONTO=@v_M  OUTPUT
```

```
PRINT 'CANTIDAD DE PEDIDOS COLOCADOS:' + STR (@v_Q)
```

```
PRINT 'MONTO PERCIBIDO:' + STR (@v_M)
```

```
GO
```

El resultado se mostrará así:

Mensajes		
CANTIDAD DE PEDIDOS COLOCADOS:		102
MONTO PERCIBIDO:		74213

5.2 MANEJO DE PROCEDIMIENTOS ALMACENADOS II

5.2.1 Anidamiento de procedimientos almacenados.

Los procedimientos almacenados se anidan cuando un procedimiento almacenado llama a otro o ejecuta código administrado mediante una referencia a una rutina, tipo o función de agregado CLR. Puede anidar hasta 32 niveles de procedimientos almacenados y referencias a código administrado. El nivel de anidamiento aumenta en uno cuando el procedimiento almacenado o la referencia de código administrado a los que se ha llamado empiezan a ejecutarse, y disminuye en uno cuando finaliza su ejecución. Si se intenta superar el límite máximo de 32 niveles de anidamiento, se producirá un error general en la cadena de llamada. El nivel actual de anidamiento de los procedimientos almacenados en ejecución se almacena en la función @@NESTLEVEL.

Ejemplo de cómo un procedimiento almacenado llama a otro.

Paso 1:

Elaborar un store procedure que devuelva el número de órdenes de compra, dirigidas a un determinado proveedor.

Paso 2:

Construir un store procedure que invoque al store procedure del paso anterior, para determinar si se elimina o no, a un proveedor, dependiendo si tiene o no, órdenes generadas.

```

/* PASO 1*/
USE VENTASCIB
GO

/* CREANDO EL PROCEDIMIENTO ALMACENADO */
CREATE PROCEDURE USP_ORDENES_PROVEEDOR
@VCODPRV CHAR (4),
@VNUMORD INT OUTPUT
AS
BEGIN
    SELECT @VNUMORD = (SELECT COUNT (*)
                        FROM TB_ORDEN_COMPRA
                        WHERE COD_PRV = @VCODPRV)

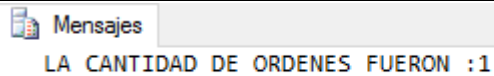
END
GO

/* EJECUCIÓN */
DECLARE @VCANT INT
EXEC USP_ORDENES_PROVEEDOR 'PR01', @VCANT OUTPUT

```

```
PRINT 'LA CANTIDAD DE ÓRDENES FUERON: '+ CONVERT (CHAR (2),@VCANT)
GO
```

El resultado es:



Mensajes

LA CANTIDAD DE ORDENES FUERON :1

```
/* PASO 2 */
USE VENTASCIB
GO

/* CREANDO EL PROCEDIMIENTO ALMACENADO */
CREATE PROCEDURE USP_ELIMINARE_PROVEEDOR
@VCODPRV CHAR (4)
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM TB_PROVEEDOR
                    WHERE @VCODPRV=COD_PRV)
        PRINT 'NO EXISTE EL PROVEEDOR INGRESADO'
    ELSE
        BEGIN
            DECLARE @VORDENES INTEGER
            EXEC USP_ORDENES_PROVEEDOR
                @VCODPRV,
                @VORDENES OUTPUT
            IF @VORDENES = 0
                PRINT 'PROVEEDOR SERA ELIMINADO'
            ELSE
                PRINT 'PROVEEDOR SE MANTIENE'
        END
    END
GO

/* EJECUCIÓN */
EXEC USP_ELIMINARE_PROVEEDOR 'PR01'
GO
```

El resultado es:



Mensajes

PROVEEDOR SE MANTIENE

5.2.2 Operaciones con procedimientos almacenados.

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)

```
USE NEGOCIOS
GO

CREATE PROCEDURE USP_INVENTARIO
AS
BEGIN
    -- 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,
                                         NOMPRODUCTO,
                                         PRECIOUNIDAD,
                                         UNIDADES EN EXISTENCIA
                                FROM COMPRAS.PRODUCTOS

    -- APERTURA DEL CURSOR
    OPEN CPRODUCTO
    -- LECTURA DE LA PRIMERA FILA DEL CURSOR
    FETCH CPRODUCTO INTO @ID, @NOMBRE, @PRECIO, @ST
    PRINT SPACE (8) + 'ID' + SPACE (6) + 'PRODUCTO' + SPACE (38) +
        'PRECIO' + SPACE (10) + 'STOCK'
    PRINT REPLICATE ('=', 90)
    WHILE ( @@FETCH_STATUS = 0)
    BEGIN
        -- IMPRIMIR
        PRINT STR (@ID) + SPACE (5) +
            @NOMBRE + SPACE (40-LEN (RTRIM (@NOMBRE))) +
            STR (@PRECIO) + SPACE (5) + STR (@ST)
        -- ACUMULAR
        SET @INV += @ST
    END
END
```

```

-- 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)
END
GO

```

Ejecutamos:

```

EXECUTE USP_INVENTARIO
GO

```

Este es el resultado:

Mensajes			
ID	PRODUCTO	PRECIO	STOCK
1	Te Dharamsala	7	39
2	Cerveza tibetana Barley	8	17
3	Sirope de regaliz	4	13
4	Especias Cajun del chef Anton	9	53
5	Mezcla Gumbo del chef Anton	9	0
6	Mermelada de grosellas de la abuela	10	120
7	Peras secas organicas del tio Bob	12	15
8	Salsa de arandanos Northwoods	17	6
9	Buey Mishi Kobe	40	29
10	Pez espada	13	31
<hr/>			
70	Cerveza Outback	6	15
71	Crema de queso Fløtemys	9	26
72	Queso Mozzarella Giovanni	14	14
73	Caviar rojo	6	101
74	Queso de soja Longlife	4	4
75	Cerveza Klosterbier Rhönbräu	3	125
76	Licor Cloudberry	7	57
77	Salsa verde original Frankfurter	5	32
<hr/>			
INVENTARIO DE PRODUCTOS: 3119			

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

```

USE NEGOCIOS
GO

CREATE PROCEDURE USP_REPORTEPEDIDOSXAÑOXEMPLEADO
@EMP INT=1
AS
BEGIN
-- 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)
END
GO

```

Ahora ejecutamos:

```

EXEC USP_REPORTEPEDIDOSXAÑOXEMPLEADO 2
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.

Mensajes	
Año:1996	
10265	1170
10277	1188
10280	596
10295	120
10300	590
10307	420
10312	1588
10313	180
10327	2175
10339	3432
10345	2899
10368	1793
10379	936
10388	1245
10392	1400
10398	2700
Importe en el año:1996 es 22432	
Año:1997	
10404	1640
10407	1155
10414	226
10422	48
10457	1584
10462	151

Consulta ejecutada correctamente.

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

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.

```

USE NEGOCIOS
GO

```

```

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

```

USE NEGOCIOS
GO

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, APEMPLEADO, FECNAC,
    DIREMPLEADO, IDISTRITO, FONOEMPLEADO, IDCARGO,
    FECONTRATA)
    ON (TARGET.IDEMPLEADO = SOURCE.IDEMPLEADO)
    WHEN MATCHED THEN

```

```

UPDATE
SET NOMEMPLEADO=@NOMBRE, APEMPELEADO=@APELLIDO,
    FECNAC=@FN, DIREMPLEADO=@DIRECCION,
    IDISTRITO=@IDDIS, FONOEMPLEADO=@FONO,
    IDCARGO=@IDCARGO, FECCONTRATA=@FC
WHEN NOT MATCHED THEN
INSERT
VALUES
    (@ID, @NOMBRE, @APELLIDO, @FN, @DIRECCION, @IDDIS,
    @FONO, @IDCARGO, @FC);
GO

```

Ejecutamos:

```

EXEC USP_ACTUALIZAEMPLEADO '1','Nancy','Davolio','10/10/80',
    'Ca Los Paujiles 330','2','6573344','2','11/11/00'
GO

```

Al hacer la consulta de la tabla empleados:

```
SELECT * FROM RRHH.EMPLEADOS;
```

Vemos que el primer registro tuvo cambios en algunos campos

Resultados		Mensajes							
	IdEmpleado	ApeEmpleado	NomEmpleado	FecNac	DirEmpleado	idDistrito	fonoEmpleado	idCargo	FecContrata
1	1	Davolio	Nancy	1980-10-10 00:00:00.000	Ca Los Paujiles 330	2	6573344	2	2000-11-11 00:00:00.000
2	2	Fuller	Andrew	1952-02-19 00:00:00.000	Av Abancay 234	1	98788766	1	2011-02-09 00:00:00.000
3	3	Leverting	Janet	1963-08-30 00:00:00.000	Av. Riva Agüero 233	4	5664555	3	2011-03-10 00:00:00.000
4	4	Peacock	Margaret	1958-09-19 00:00:00.000	Calle las flores 411	2	980523344	1	2011-03-11 00:00:00.000
5	5	Buchanan	Steven	1955-03-04 00:00:00.000	Av. Brasil 222	3	6776566	3	2011-04-05 00:00:00.000
6	6	Suyama	Michael	1963-07-02 00:00:00.000	Calle Zafiro 344	1	8997877	4	2011-04-10 00:00:00.000
7	7	King	Robert	1960-05-29 00:00:00.000	Jr Puni 322	1	99876677	2	2011-05-04 00:00:00.000
8	8	Callahan	Laura	1958-01-09 00:00:00.000	Dinthilac 123	2	90098999	4	2011-06-10 00:00:00.000
9	9	Dodsworth	Anne	1969-07-02 00:00:00.000	Av Los fresnos 2334	3	98877888	1	2011-07-10 00:00:00.000

Si ejecutamos nuevamente con otros valores en los parámetros:

```

EXEC USP_ACTUALIZAEMPLEADO '10','Amy','Stewar','10/10/85',
    'Ca Los Nogales 666','3','6141717','1','11/11/05'
GO

```

Nuevamente hacemos la consulta:

```
SELECT * FROM RRHH.EMPLEADOS;
```

Vemos que se agregó un nuevo registro, quiere decir que está funcionando perfectamente el Merge del procedimiento.

Resultados		Mensajes							
	IdEmpleado	ApeEmpleado	NomEmpleado	FecNac	DirEmpleado	idDistrito	fonoEmpleado	idCargo	FecContrata
1	1	Davolio	Nancy	1980-10-10 00:00:00.000	Ca Los Paujiles 330	2	6573344	2	2000-11-11 00:00:00.000
2	2	Fuller	Andrew	1952-02-19 00:00:00.000	Av Abancay 234	1	98788766	1	2011-02-09 00:00:00.000
3	3	Leverling	Janet	1963-08-30 00:00:00.000	Av. Riva Agüero 233	4	5664555	3	2011-03-10 00:00:00.000
4	4	Peacock	Margaret	1958-09-19 00:00:00.000	Calle las flores 411	2	980523344	1	2011-03-11 00:00:00.000
5	5	Buchanan	Steven	1955-03-04 00:00:00.000	Av. Brasil 222	3	6776566	3	2011-04-05 00:00:00.000
6	6	Suyama	Michael	1963-07-02 00:00:00.000	Calle Zafiro 344	1	8997877	4	2011-04-10 00:00:00.000
7	7	King	Robert	1960-05-29 00:00:00.000	Jr Puni 322	1	99876677	2	2011-05-04 00:00:00.000
8	8	Callahan	Laura	1958-01-09 00:00:00.000	Dinhtilac 123	2	90098999	4	2011-06-10 00:00:00.000
9	9	Dodsworth	Anne	1969-07-02 00:00:00.000	Av Los fresnos 2334	3	98877888	1	2011-07-10 00:00:00.000
10	10	Amy	Stewar	1985-10-10 00:00:00.000	Ca Los Nogales 666	3	6141717	1	2005-11-11 00:00:00.000

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 TSQL 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.

5.3 MANEJO DE FUNCIONES DE USUARIO

5.3.1 Funciones del sistema.

Funciones para el manejo de fechas

Función	Descripción
DATEADD	<p>Devuelve un valor <i>date</i> con el intervalo <i>number</i> especificado, agregado a un valor <i>datepart</i> especificado de ese valor <i>date</i>.</p> <p>DATEADD (datepart , number , date)</p> <pre> 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 </pre>

DATEDIFF	<p>Devuelve el número de límites <i>datepart</i> de fecha y hora entre dos fechas especificadas.</p> <p>DATEDIFF (<i>datepart</i> , <i>startdate</i> , <i>enddate</i>)</p> <p>SET DATEFORMAT DMY DECLARE @FECHAINICIAL DATE = '01-08-2011'; DECLARE @FECHAFINAL DATE = '01-09-2011'; SELECT DATEDIFF(DAY, @FECHAINICIAL,@FECHAFINAL) AS 'DURACION'</p>
DATENAME	<p>Devuelve una cadena de caracteres que representa el <i>datepart</i> especificado de la fecha especificada.</p> <p>DATENAME (<i>datepart</i> , <i>date</i>)</p> <p>SELECT DATENAME(MONTH, GETDATE()) AS 'MES';</p>
DATEPART	<p>Devuelve un entero que representa el <i>datepart</i> especificado del</p>
	<p><i>date</i> especificado.</p> <p>DATEPART (<i>datepart</i> , <i>date</i>)</p> <p>SELECT DATEPART(MONTH, GETDATE()) AS 'MES';</p>
DAY	<p>Devuelve un entero que representa la parte del día datepart de la fecha especificada.</p> <p>SELECT DAY('01/9/2011') AS 'DÍA DEL MES';</p>
GETDATE	<p>Devuelve la fecha del sistema</p> <p>SELECT GETDATE() 'FECHA DEL SISTEMA';</p>
MONTH	<p>Devuelve un entero que representa el mes de <i>date</i> especificado. MONTH devuelve el mismo valor que DATEPART (<i>month</i>, <i>date</i>).</p> <p>SELECT MONTH(GETDATE()) AS 'MES DE LA FECHA DE SISTEMA';</p>
YEAR	<p>Devuelve un entero que representa el año de <i>date</i> especificado. YEAR devuelve el mismo valor que DATEPART (<i>year</i>, <i>date</i>).</p> <p>SELECT YEAR(GETDATE()) AS 'AÑO DE LA FECHA DE SISTEMA';</p>

Funciones para el manejo de cadenas

LEFT	<p>Devuelve la parte izquierda de una cadena de caracteres con el número de caracteres especificado.</p> <p>LEFT (<i>character_expression</i> , <i>integer_expression</i>)</p>
-------------	---

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)

Ejemplo:

```
-- 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
```

El resultado es:

Resultados Mensajes	
	CADENA RESULTANTE
1	BASE

Funciones de conversión

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

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])
STR	Devuelve datos de caracteres convertidos de datos numéricos. STR (float_expression [, length [, decimal]])

Ejemplo:

Vea los datos como se muestra:

```
USE NEGOCIOS
GO

SELECT DISTINCT
    NOMPRODUCTO AS NOMBRE,
    PRECIOUNIDAD AS 'PRECIO UNITARIO',
    UNIDADESENEXISTENCIA AS 'STOCK'
FROM COMPRAS.PRODUCTOS
GO
```

El resultado es:

	NOMBRE	PRECIO UNITARIO	STOCK
1	Algas Konbu	2	24
2	Arenque ahumado	4	5
3	Arenque blanco del noroeste	10	10
4	Arenque salado	5	95
5	Azúcar negra Malacca	8	27
6	Barras de pan de Escocia	5	6
7	Bollos de pan de Wimmer	14	22
8	Bollos de Sir Rodneys	4	3
9	Buey Mishi Kobe	40	29
10	Café de Malasia	19	17

Ahora ejecutemos con las funciones de conversión:

```
USE NEGOCIOS
```

GO

SELECT DISTINCT

```
CAST (NOMPRODUCTO AS CHAR (5)) AS NOMBRE_CONV,
CONVERT (SMALLMONEY, PRECIOUNIDAD, 1) AS 'PRECIO CONV',
STR (UNIDADES ENEXISTENCIA) AS 'STOCK CONV'
```

FROM COMPRAS.PRODUCTOS

GO

El resultado es:

	NOMBRE_CONV	PRECIO CONV	STOCK CONV
1	Algas	2,00	24
2	Arenq	4,00	5
3	Arenq	5,00	95
4	Arenq	10,00	10
5	Azúca	8,00	27
6	Barra	5,00	6
7	Bollo	4,00	3
8	Bollo	14,00	22
9	Buey	40,00	29
10	Cafe	19,00	17

Ejemplo de conversion de fechas:

Ejecutando este script, se muestra los datos sin transformar:

USE NEGOCIOS

GO

```
SELECT NOMEMPLEADO,
       APEEMPLEADO,
       FECNAC,
       FECCONTRATA
```

FROM RRHH.EMPLEADOS

GO

Se muestra así:

	NomEmpleado	ApeEmpleado	FecNac	FecContrata
1	Nancy	Davolio	1980-10-10 00:00:00.000	2000-11-11 00:00:00.000
2	Andrew	Fuller	1952-02-19 00:00:00.000	2011-02-09 00:00:00.000
3	Janet	Leverling	1963-08-30 00:00:00.000	2011-03-10 00:00:00.000
4	Margaret	Peacock	1958-09-19 00:00:00.000	2011-03-11 00:00:00.000
5	Steven	Buchanan	1955-03-04 00:00:00.000	2011-04-05 00:00:00.000
6	Michael	Suyama	1963-07-02 00:00:00.000	2011-04-10 00:00:00.000
7	Robert	King	1960-05-29 00:00:00.000	2011-05-04 00:00:00.000
8	Laura	Callahan	1958-01-09 00:00:00.000	2011-06-10 00:00:00.000
9	Anne	Dodsworth	1969-07-02 00:00:00.000	2011-07-10 00:00:00.000
10	Stewar	Amy	1985-10-10 00:00:00.000	2005-11-11 00:00:00.000

Ahora, ejecutando el script con conversión, utilizando estilos:

```
USE NEGOCIOS
GO

SELECT NOMBRE AS NOMBRE,
       APELLIDO AS APELLIDO,
       CONVERT (VARCHAR (20), FECNAC, 113) AS FECHA_NAC,
       CONVERT (VARCHAR (12), FECCONTRATA, 106) AS FECHA_CON
FROM RRHH.EMPLEADOS
GO
```

El resultado sería:

	NOMBRE	APELLIDO	FECHA_NAC	FECHA_CON
1	Nancy	Davolio	10 Oct 1980 00:00:00	11 Nov 2000
2	Andrew	Fuller	19 Feb 1952 00:00:00	09 Feb 2011
3	Janet	Leverling	30 Ago 1963 00:00:00	10 Mar 2011
4	Margaret	Peacock	19 Sep 1958 00:00:00	11 Mar 2011
5	Steven	Buchanan	04 Mar 1955 00:00:00	05 Abr 2011
6	Michael	Suyama	02 Jul 1963 00:00:00	10 Abr 2011
7	Robert	King	29 May 1960 00:00:00	04 May 2011
8	Laura	Callahan	09 Ene 1958 00:00:00	10 Jun 2011
9	Anne	Dodsworth	02 Jul 1969 00:00:00	10 Jul 2011
10	Stewart	Amy	10 Oct 1985 00:00:00	11 Nov 2005

5.3.2 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

5.3.2.1 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.

Sintaxis:

```
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

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

Utilizando la función:

```
PRINT DBO.PRECIOPROMEDIO ()  
GO
```

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

```
USE NEGOCIOS  
GO  
  
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
```

Utilizando la función:

```
PRINT DBO.PEDIDOSEMPLEADO (4)
GO
```

5.3.2.2 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 queries 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 País.

```
USE NEGOCIOS
GO

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

Se ejecuta:

```
SELECT * FROM DBO.CLIENTES () WHERE PAIS='CHILE'
GO
```

El resultado es:

	CODIGO	CLIENTE	DIRCLIENTE	PAIS
1	BOTTM	Bottom-Dollar Markets	23 Tsawassen Blvd.	Chile
2	FISSA	FISSA Fabrica Inter. Salchichas S.A.	C/ Morazarzal, 86	Chile
3	FOLKO	Folk och fä HB	Åkergatan 24	Chile
4	ISLAT	Island Trading	Garden House\r\nCrowther Way	Chile
5	MAISD	Maison Dewey	Rue Joseph-Bens 532	Chile
6	OTTIK	Ottiles Käseladen	Mehrheimerstr. 369	Chile
7	PERIC	Pericles Comidas clasicas	Calle Dr. Jorge Cash 321	Chile
8	REGGC	Reggiani Caseifici	Strada Provinciale 124	Chile
9	RICSU	Richter Supermarkt	Grenzacherweg 237	Chile
10	SPECD	Specialites du monde	25, rue Lauriston	Chile
11	THEBI	The Big Cheese	89 Jefferson Way\r\nSuite 2	Chile

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.

USE NEGOCIOS

GO

CREATE FUNCTION DBO.PEDIDOSAÑO (@Y INT)

RETURNS TABLE

AS

```

    RETURN (SELECT PC.IDPEDIDO AS 'PEDIDO',
                  FECHAPEDIDO,
                  NOMPRODUCTO,
                  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

Ejecutamos la función:

SELECT * FROM DBO.PEDIDOSAÑO (2010)

GO

El resultado es:

	PEDIDO	FECHAPEDIDO	NOMPRODUCTO	PRECIO	CANTIDAD
1	11068	2010-09-04 00:00:00.000	Col fermentada Rössle	45	8
2	11068	2010-09-04 00:00:00.000	Cafe de Malasia	46	36
3	11068	2010-09-04 00:00:00.000	Salsa verde original Frankfurter	13	28
4	11069	2010-10-04 00:00:00.000	Licor verde Chartreuse	18	20
5	11070	2010-10-05 00:00:00.000	Te Dharamsala	18	40

5.3.2.3 Funciones multisentencia

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:

Sintaxis:

```
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.

```
USE NEGOCIOS
GO

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

Ejecuta:

```
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.

```
USE NEGOCIOS
GO
```

```

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

```

Utilizamos la función:

```

SELECT * FROM DBO.REPORTEVENTAS (2010)
GO

```

El resultado es:

	ID	NOMBRE	CANTIDAD	MONTO
1	1	Davolio	3	870
2	2	Fuller	6	2140
3	4	Peacock	4	5154
4	8	Callahan	3	2380

Se muestra un ejemplo, de cómo se calculan las ventas por empleado, en la BD Northwind.

```

USE NORTHWIND
GO

CREATE FUNCTION DBO.VENTASEMPLEADOS ()
RETURNS @REPORTE TABLE (IDEMPLEADO INT,
                          NOMBREEMPLEADO VARCHAR (50),
                          VENTASTOTALES MONEY)

AS
BEGIN
    DECLARE @IDEMP INT, @NOMBRE VARCHAR (20),
            @APELLIDO VARCHAR (30)

```

```

DECLARE @VALOR MONEY
DECLARE CUR_EMPLEADOS CURSOR FOR
                                SELECT EMPLOYEEID,
                                       FIRSTNAME,
                                       LASTNAME
                                FROM EMPLOYEES

OPEN CUR_EMPLEADOS
FETCH NEXT FROM CUR_EMPLEADOS
                                INTO @IDEMP, @NOMBRE, @APELLIDO
WHILE @@FETCH_STATUS=0
BEGIN
    SELECT @VALOR = SUM (OD.UNITPRICE*OD.QUANTITY)
    FROM ORDERS O INNER JOIN [ORDER DETAILS] OD
    ON O.ORDERID=OD.ORDERID
    WHERE O.EMPLOYEEID=@IDEMP
    INSERT INTO @REPORTE
    VALUES
    (@IDEMP, @NOMBRE+ ' ' +@APELLIDO, @VALOR)
    FETCH NEXT FROM CUR_EMPLEADOS
    INTO @IDEMP, @NOMBRE, @APELLIDO
END
/* SE LIBERA CURSOR*/
CLOSE CUR_EMPLEADOS
DEALLOCATE CUR_EMPLEADOS
RETURN

END
GO

```

Ejecutar el uso de la función:

```

SELECT * FROM VENTASEMPLEADOS ()
GO

```

El resultado es:

Resultados		Mensajes	
	Idempleado	NombreEmpleado	VentasTotales
1	1	Nancy Davolio	202143,71
2	2	Andrew Fuller	177749,26
3	3	Janet Leverling	213051,30
4	4	Margaret Peacock	250187,45
5	5	Steven Buchanan	75567,75
6	6	Michael Suyama	78198,10
7	7	Robert King	141295,99
8	8	Laura Callahan	133301,03
9	9	Anne Dodsworth	82964,00

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

5.4 DESENCADENADORES

Los desencadenadores o 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.

Un trigger (o desencadenador) es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos.

SQL Server proporciona los siguientes tipos de triggers:

- Trigger DML, se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.
- Trigger DDL, se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP de Transact-SQL, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

5.4.1 Desencadenadores DML.

Los trigger DML se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.

Los disparadores pueden ayudar 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.

La sintaxis general de un trigger es la siguiente:

```
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.

Creando el trigger:

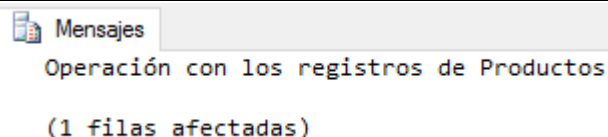
```
USE NEGOCIOS
GO

CREATE TRIGGER TX_PRODUCTOS
ON COMPRAS.PRODUCTOS FOR INSERT, UPDATE, DELETE
AS
PRINT 'OPERACIÓN CON LOS REGISTROS DE PRODUCTOS';
GO
```

Insertando un nuevo producto:

```
INSERT COMPRAS.PRODUCTOS
VALUES
('97','MIEL DE ABEJA','1','1','01 LITRO','20','10','0')
GO
```

El resultado será un nuevo registro en la tabla, mostrando el mensaje implementado en el trigger:



Mensajes

Operación con los registros de Productos

(1 filas afectadas)

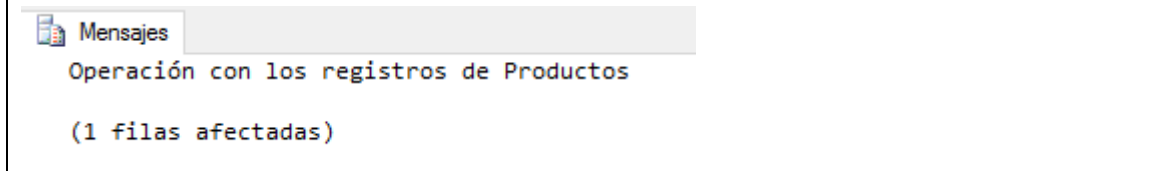
Cuando ejecute una actualización, se modificará el dato, mostrando el mensaje definido en el trigger.

```
UPDATE COMPRAS.PRODUCTOS
SET PRECIOUNIDAD = PRECIOUNIDAD + 10
WHERE IDPRODUCTO = '97'
GO
```



Lo mismo sucederá cuando realice una eliminación

```
DELETE FROM COMPRAS.PRODUCTOS
WHERE IDPRODUCTO = '97'
GO
```

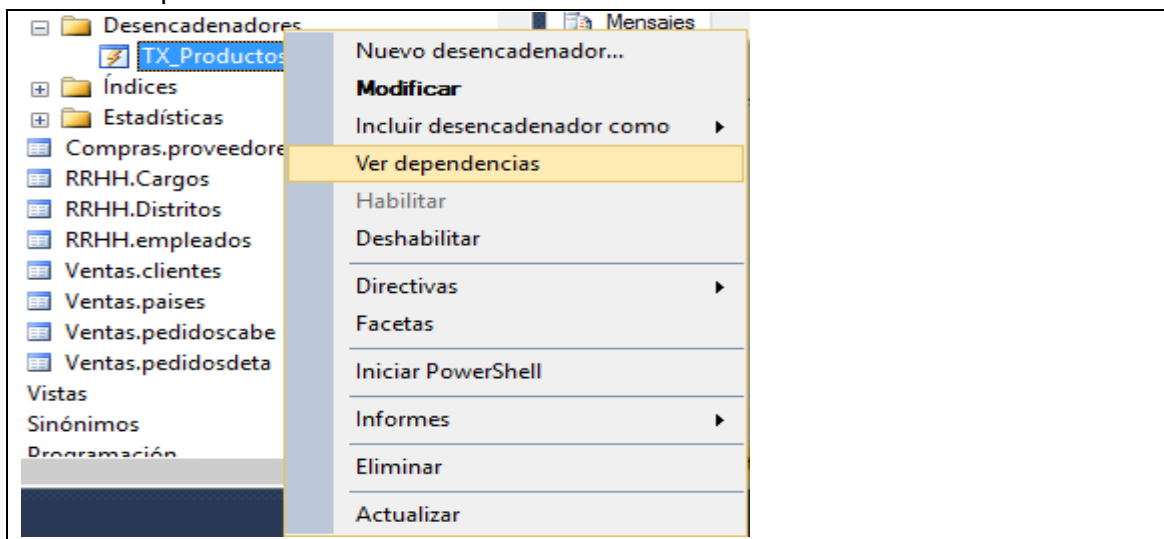


Para ver la dependencia del trigger en relación de la tabla.

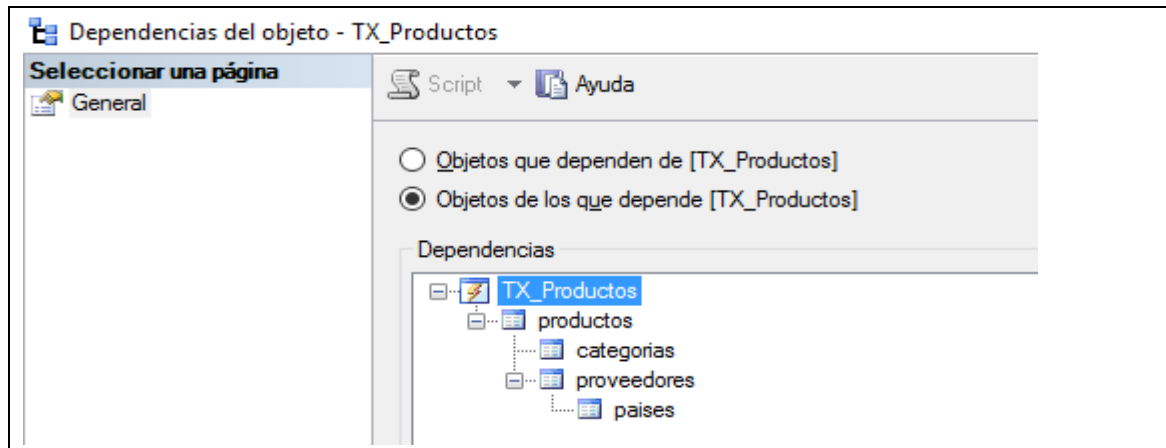
En el explorador de objetos, en la base de datos Negocios, desplegar la tabla Compras.productos, luego desencadenadores, tal como se muestra en la figura:



Hacer click secundario sobre el trigger TX_Productos, recientemente creado, luego click en dependencias.



Allí podrá ver las dependencias que tiene el trigger.



¿Que són las tablas INSERTED y DELETED?

Las instrucciones de desencadenadores DML utilizan dos tablas especiales: la tabla deleted y la tabla inserted. SQL Server crea y administra automáticamente ambas tablas. Puede utilizar estas tablas temporales residentes en memoria para probar los efectos de determinadas modificaciones de datos y para establecer condiciones para las acciones de los desencadenadores DML. No puede modificar directamente los datos de estas tablas ni realizar en ellas operaciones de lenguaje de definición de datos (DDL), como CREATE INDEX.

En los desencadenadores DML, las tablas inserted y deleted se utilizan principalmente para realizar las siguientes tareas:

- Ampliar la integridad referencial entre tablas.
- Insertar o actualizar datos de tablas base subyacentes a una vista.
- Comprobar errores y realizar acciones en función del error.
- Conocer la diferencia entre el estado de una tabla antes y después de realizar una modificación en los datos, y actuar en función de dicha diferencia.

La tabla deleted almacena copias de las filas afectadas por las instrucciones DELETE y UPDATE. Durante la ejecución de una instrucción DELETE o UPDATE, las filas se eliminan de la tabla del desencadenador y se transfieren a la tabla deleted. La tabla deleted y la tabla del desencadenador no suelen tener filas en común.

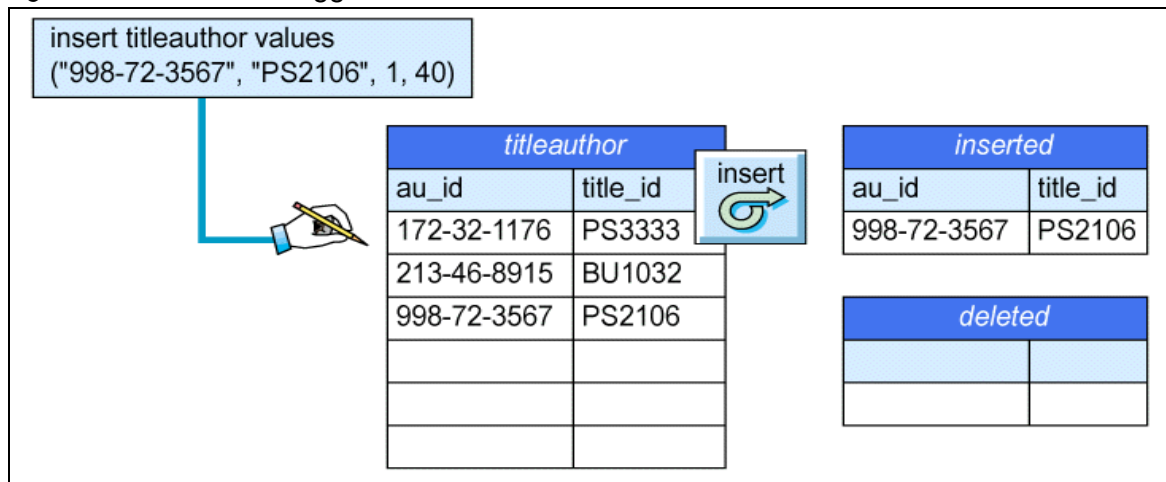
La tabla inserted almacena copias de las filas afectadas durante las instrucciones INSERT y UPDATE. Durante una transacción de inserción o actualización, se agregan nuevas filas a la tabla inserted y a la tabla del desencadenador. Las filas de la tabla inserted son copias de las nuevas filas de la tabla del desencadenador.

Una transacción de actualización es similar a una operación de eliminación seguida de una operación de inserción; primero, se copian las filas antiguas en la tabla deleted y luego se copian las filas nuevas en la tabla del desencadenador y en la tabla inserted.

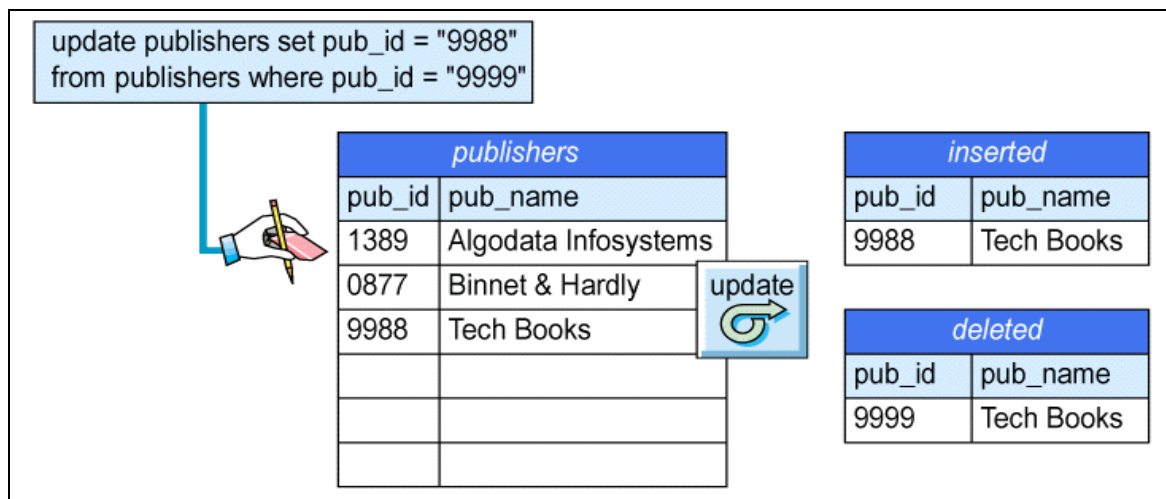
Cuando establezca condiciones para el desencadenador, utilice las tablas inserted y deleted correspondientes a la acción que lo activó. Aunque no se produce ningún error al hacer referencia a la tabla deleted cuando se prueba una instrucción INSERT, o bien al hacer referencia a la tabla inserted cuando se prueba una instrucción DELETE, estas tablas de prueba del desencadenador no contendrán filas en estos casos.

SQL Server 2014 no permite referencias de las columnas text, ntext o image en las tablas inserted y deleted para desencadenadores AFTER. Sin embargo, estos tipos de datos se incluyen únicamente por motivos de compatibilidad con versiones anteriores. El almacenamiento preferido para datos de gran tamaño es el uso de tipos de datos varchar (max), nvarchar (max) y varbinary (max). Tanto los desencadenadores AFTER como INSTEAD OF admiten los datos varchar (max), nvarchar (max) y varbinary (max) en las tablas inserted y deleted

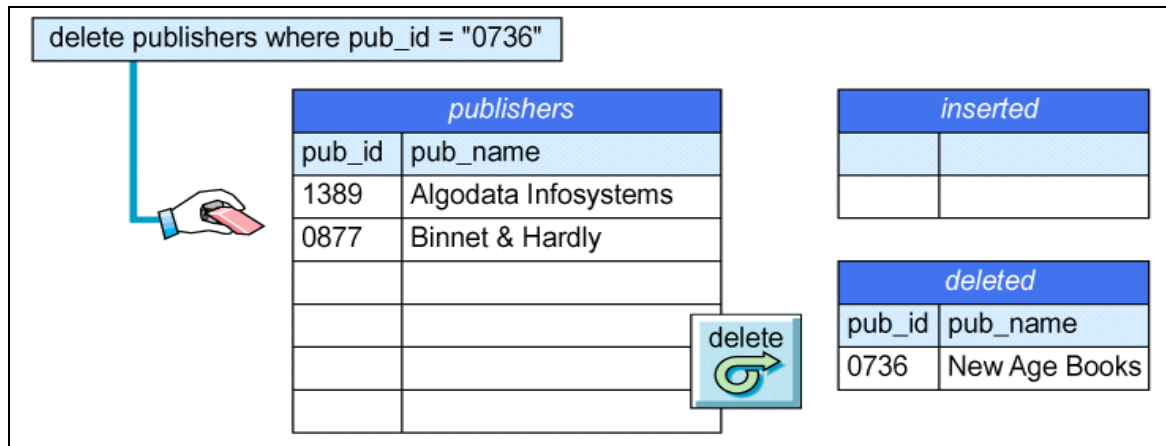
¿Cómo funciona un trigger al realizar un INSERT?



¿Cómo funciona un trigger al realizar un UPDATE?



¿Cómo funciona un trigger al realizar un UPDATE?



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.

Este ejemplo muestra un trigger que se dispara cuando se inserta una factura con estado '1' (pendiente). La regla de negocio que se valida es que el cliente tenga un crédito (de tipo '1'), para que se le permita tener una factura pendiente.

```
USE VENTASCIB
GO

CREATE TRIGGER TR_CLIENTES
ON TB_FACTURA
FOR INSERT
AS
    IF (SELECT EST_FAC FROM INSERTED) ='1'
        BEGIN TRAN
            IF (SELECT TIP_CLI
                FROM TB_CLIENTE C, INSERTED I
                WHERE C.COD_CLI=I.COD_CLI) ='2'
                BEGIN
                    PRINT 'NO SE PUEDE DAR CREDITO A ESTE CLIENTE'
                    ROLLBACK TRANSACTION
                END
            ELSE
                COMMIT TRANSACTION
        GO
```

Ahora, realizamos la inserción de una nueva factura:

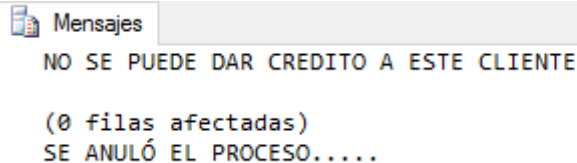
```
BEGIN TRY
    INSERT TB_FACTURA
    VALUES
        ('FA092','10/10/15','C002','11/11/15','1','V02','0')
END TRY
```

```

BEGIN CATCH
    IF @@ERROR <> 0
        PRINT 'SE ANULÓ EL PROCESO.....'
END CATCH
GO

```

No se realiza la transacción porque el cliente C002 es de tipo 2.



Mensajes

NO SE PUEDE DAR CREDITO A ESTE CLIENTE

(0 filas afectadas)

SE ANULÓ EL PROCESO.....


Sin embargo, si se hace la transacción al cliente C001, si es efectivo.

```

BEGIN TRY
    INSERT TB_FACTURA
    VALUES
        ('FA092','10/10/15','C001','11/11/15','1','V02','0')
END TRY
BEGIN CATCH
    IF @@ERROR <> 0
        PRINT 'SE ANULÓ EL PROCESO.....'
END CATCH
GO

```

El resultado será:



Mensajes

(1 filas afectadas)

Disparador de Eliminación

Este ejemplo muestra un trigger de eliminación, cuya finalidad es que al eliminar un vendedor, se verifique si tiene facturas generadas, si es así, que se envíe un mensaje indicando que no se puede eliminar dicho vendedor, pero de lo contrario, se asigne un valor nulo al distrito donde estaba asignado dicho vendedor.

```

USE VENTASCIB
GO

CREATE TRIGGER TX_ELIMINARVND
ON TB_VENDEDOR
FOR DELETE
AS
    IF NOT EXISTS (SELECT * FROM TB_FACTURA
                   WHERE COD_VEN = (SELECT DELETED.COD_VEN
                                     FROM DELETED))

```

```

        BEGIN
            PRINT 'ELIMINACIÓN EFECTIVA'
            COMMIT TRANSACTION
        END
    ELSE
        BEGIN
            ROLLBACK TRANSACTION
            PRINT 'NO SE PUEDE ELIMINAR AL VENDEDOR'
            PRINT 'PORQUE TIENE FACTURAS GENERADAS'
        END
    GO

```

Disparadores de Actualización

Finalmente, este ejemplo muestra un trigger de actualización. Cuando una orden se anule, es decir, su estado pase a valor '3' todos sus detalles se deben eliminar.

```

USE VENTASCIB
GO

CREATE TRIGGER TR_ORDEN_COMPRA
ON TB_ORDEN_COMPRA
FOR UPDATE
AS
    IF EXISTS (SELECT EST_OCO = '3' FROM INSERTED)
        DELETE FROM TB_DETALLE_COMPRA
        WHERE NUM_OCO = (SELECT NUM_OCO FROM INSERTED)
GO

```


Si ejecuta esta transacción:

```

UPDATE TB_ORDEN_COMPRA
SET EST_OCO = '3'
WHERE NUM_OCO = 'OC003'
GO

```

El resultado será la actualización en la tabla TB_ORDEN_COMPRA, y la eliminación de los registros de la tabla TB_DETALLE_COMPRA.

 Mensajes

```

(1 filas afectadas)

(1 filas afectadas)

```

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.

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.

```
USE NEGOCIOS
GO
```

```
--PRIMERO, CREAR LA TABLA CIENTESBAK
SELECT * INTO VENTAS.CLIENTESBAK
FROM VENTAS.CLIENTES
WHERE IDCLIENTE = 'XXXX'
GO
```

```
-- CREAR UNA VISTA QUE LISTE LAS COLUMNAS DE CLIENTES.
```

```
CREATE VIEW INSTEADVIEW
AS
SELECT IDCLIENTE,
       NOMCLIENTE,
       DIRCLIENTE,
       IDPAIS,
       FONOCIENTE
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, FONOCIENTE
    FROM INSERTED
```



```
END  
GO
```

5.4.2 Desencadenadores DDL.

Los triggers de tipo DDL se disparan cuando se ejecuta una sentencia CREATE, ALTER y/o DROP, ya sea de un objeto a nivel de servidor o de base de datos. Estos triggers contienen código para almacenar en tablas de auditoría, las definiciones realizadas por el usuario, por lo tanto, es útil para hacer auditoría sobre los objetos de un servidor o de una base de datos.

También hay que tomar en cuenta que la ejecución del trigger puede tener impacto en el desempeño de la sentencia, ya que añade procesamiento.

Hay dos tipos de triggers DDL:

- A nivel de servidor. Se ejecuta a nivel de servidor después de la sentencia CREATE, ALTER y/o DROP. Permite hacer auditoría sobre objetos, a nivel de servidor como un Login, un Endpoint o una base de datos.
- A nivel de base de datos. Se ejecuta a nivel de base de datos después de la sentencia CREATE, ALTER y/o DROP. Permite hacer auditoría sobre objetos, a nivel de base de datos como un usuario, una tabla o una función.

El uso de un desencadenador DDL con ámbito de servidor:

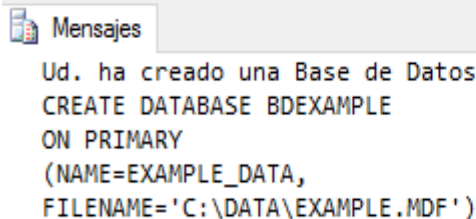
En el siguiente ejemplo se utiliza un desencadenador DDL para imprimir un mensaje en el caso de ejecutar el evento CREATE DATABASE que se produce en la instancia del servidor actual, y utiliza la función EVENTDATA para recuperar el texto de la instrucción de Transact-SQL correspondiente.

```
USE MASTER  
GO  
  
IF EXISTS (SELECT * FROM SYS.SERVER_TRIGGERS  
           WHERE NAME = 'DDL_TRIGGER_BD')  
    DROP TRIGGER DDL_TRIGGER_BD  
    ON ALL SERVER;  
GO  
  
CREATE TRIGGER DDL_TRIGGER_BD  
ON ALL SERVER  
FOR CREATE_DATABASE  
AS  
    DECLARE @MSG NVARCHAR (MAX)  
    PRINT 'UD. HA CREADO UNA BASE DE DATOS'  
    SELECT @MSG=EVENTDATA ().VALUE  
    ('(/EVENT_INSTANCE/TSQLCOMMAND/COMMANDTEXT) [1]', 'NVARCHAR (MAX)')  
    PRINT @MSG  
GO
```

Ahora comprobamos su funcionamiento:

```
CREATE DATABASE BDEXAMPLE
ON PRIMARY
(NAME=EXAMPLE_DATA,
FILENAME='C:\DATA\EXAMPLE.MDF')
GO
```

Aparecerá este resultado:



Mensajes

Ud. ha creado una Base de Datos
CREATE DATABASE BDEXAMPLE
ON PRIMARY
(NAME=EXAMPLE_DATA,
FILENAME='C:\DATA\EXAMPLE.MDF')

El uso de un desencadenador DDL con ámbito de base de datos

En el siguiente ejemplo se utiliza un desencadenador DDL para evitar que se eliminen cualquier tabla de una base de datos.

Primero, se crea la tabla TBMESSAGE en una base de datos.

```
USE BDEXAMPLE
GO

CREATE TABLE TBMESSAGE
(
ID INT IDENTITY,
MSG VARCHAR (100))
GO
```

Ahora se crea el trigger DDL:

```
--VERIFICA LA EXISTENCIA DEL TRIGGER
IF EXISTS (SELECT * FROM sys.triggers
WHERE parent_class = 0 AND name = 'safety')
DROP TRIGGER SAFETY
ON DATABASE;
GO

--CREA EL TRIGGER
CREATE TRIGGER SAFETY
ON DATABASE
FOR DROP_TABLE
AS
RAISERROR ('UD. DEBE DESACTIVAR EL TRIGGER "SAFETY" PARA ELIMINAR
UNA TABLA!',10, 1)
ROLLBACK
GO
```

Luego, comprobamos el funcionamiento del trigger impidiendo eliminar tabla alguna de esta base de datos.

```
USE BDEXAMPLE
GO

BEGIN TRY
    DROP TABLE TBMESSAGE
END TRY
BEGIN CATCH
    IF @@ERROR <> 0
        PRINT 'OPERACIÓN ABORTADA'
END CATCH
GO
```

El resultado se muestra.

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 si se crea un disparador en una tabla del sistema, el disparador no se utilizará.

Resumen

- Un cursor es una variable que nos permite recorrer con un conjunto de resultados obtenidos a través de una sentencia SELECT fila por fila.
- 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 queries 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:
 - 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
- 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.
- 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. Esta 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. Esta 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



ADMINISTRACIÓN DE BASE DE DATOS 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, así como la definición y programación de alertas y tareas que deben ejecutarse en forma periódica.

TEMARIO

6.1 Tema 13 : Seguridad y restauración en SQL Server

- 6.1.1 : Copia de seguridad en SQL Server
- 6.1.1.1 : Tipos de Backup
- 6.1.2 : Restaurando una copia de seguridad
- 6.1.3 : Base de datos Snapshots
- 6.1.3.1 : Creando una DB Snapshot
- 6.1.3.2 : Utilizar snapshot para revertir cambios

6.2 Tema 14 : Automatizar tareas en SQL Server

- 6.2.1 : Creando tareas
- 6.2.2 : Creando alertas
- 6.2.3 : Creación de planes de mantenimiento

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 SEGURIDAD Y RESTAURACIÓN 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.

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.

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 ocurran?

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?

6.1.1 Copia de seguridad en SQL Server.

SQL Server tiene tres modelos de recuperación de base de datos: simple, completa y de registro masivo. Cada modelo conserva los datos, en caso de un error del servidor, pero presentan diferencias clave en el modo en que SQL Server recupera los datos.

Puede establecer o cambiar en cualquier momento su modelo de recuperación, pero debe planear un modelo de recuperación cuando crea una base de datos.

MODELO DE RECUPERACIÓN	DESCRIPCIÓN
Simple	Usa copias completas o diferenciales de la base de datos. Trunca los registros de

	transacciones.
Completa	Incluye copias de seguridad de la base de datos y del registro de transacciones.
Registro masivo	Incluye copias de seguridad de la base de datos y del registro de transacciones, pero usa menos espacio de registro para ciertas operaciones de carga masiva.

Modelo de recuperación completo

Se podrá usar el modelo de recuperación completa cuando la recuperación completa de los medios dañados es la prioridad máxima. Este modelo usa copias de la base de datos y toda la información del registro para restaurar la base de datos.

SQL Server registra todos los cambios de la base de datos, incluso las operaciones masivas y las creaciones de índices. En caso de que los registros en sí, no estén dañados, SQL Server puede recuperar todos los datos, excepto las transacciones que realmente estaban en proceso en el momento del error.

Dado que se registran todas las transacciones, la recuperación se puede llevar a cabo en cualquier momento. Además, SQL Server admite la inserción de marcas con nombre en el registro de transacciones para permitir la recuperación hasta esa marca concreta.

Dado que las marcas de registros de transacción ocupan espacio en el registro, sólo deben usarse para transacciones que tengan una función importante en la estrategia de recuperación de la base de datos. La limitación principal de este modelo es el gran tamaño de los archivos de registro y los costos de almacenamiento y de rendimiento resultantes.

Modelo de recuperación simple

Normalmente se usa el modelo de recuperación simple para bases de datos pequeñas o en las que las modificaciones de datos no son frecuentes. Este modelo usa copias completas o diferenciales de la base de datos, y la recuperación se limita a la restauración de la base de datos hasta el momento en el que se efectuó la última copia de seguridad. Todos los cambios realizados después de la copia de seguridad se perderán y deberán aplicarse de nuevo. El beneficio principal de este modelo es que ocupa menos espacio de almacenamiento para los registros y es más sencillo de implementar.

Modelo de recuperación de carga masiva

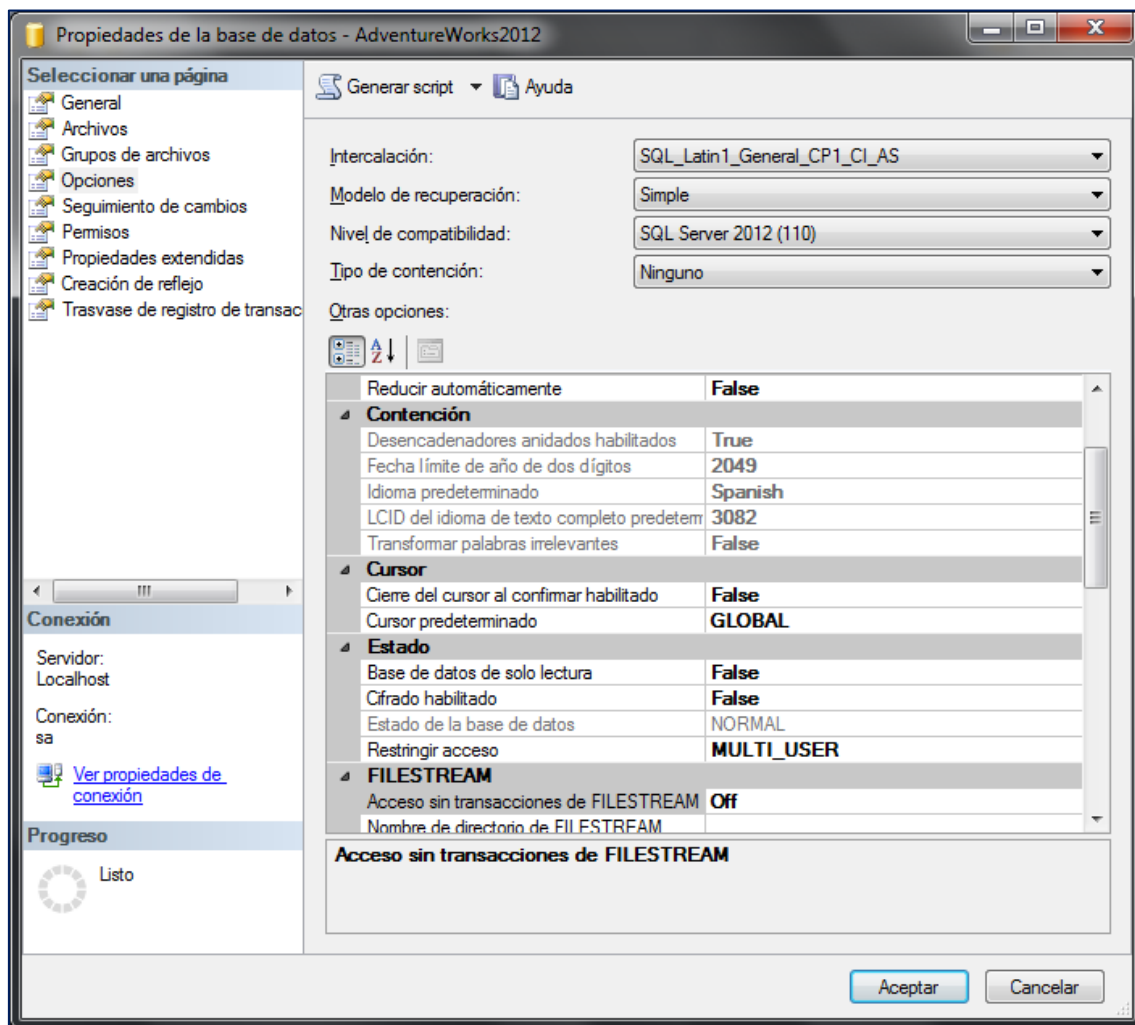
Es similar al modelo de recuperación completo, la recuperación de registro masivo usa tanto las copias de seguridad de bases de datos como las copias de seguridad del registro para volver a crear una base de datos. Sin embargo, el modelo de recuperación de registro masivo usa menos espacio del registro para las operaciones: CREATE INDEX, operaciones de carga masiva, SELECT INTO, WRITETEXT y UPDATETEXT. Estas operaciones sólo afectan al registro como bits en magnitudes en lugar de almacenar detalles de las operaciones en el registro.

Para conservar los cambios para una operación de carga masiva completa, las magnitudes que se marcan como modificadas también se almacenan en el registro. Como resultado de almacenar sólo el resultado final de varias operaciones, el registro suele ser más reducido y las operaciones masivas suelen ejecutarse más rápidamente.

Mediante el uso de este modelo pueden restaurarse todos los datos, pero una desventaja es que no es posible la restauración de sólo una parte de una copia de seguridad, como la restauración hasta una marca concreta.

¿Cómo se configura el Modelo de recuperación?

La configuración del modelo de recuperación se lleva a cabo, a través de la página Opciones de la ventana de propiedades, de cada base de datos.



También se puede utilizar la sentencia de comando ALTER DATABASE

```
ALTER DATABASE ADVENTUREWORKS2014
SET RECOVERY FULL
GO
```

```
ALTER DATABASE ADVENTUREWORKS2014
SET RECOVERY SIMPLE
```

```
GO
```

```
ALTER DATABASE ADVENTUREWORKS2014  
SET RECOVERY BULK_LOGGED  
GO
```

6.1.1.1 Tipos de Backup.

Realizar un backup a una base de datos es diferente a realizar backups sobre archivos en un servidor. Las bases de datos contienen logs de transacciones y en algunos casos, también es necesario realizar un backup sobre ellos.

SQL Server incluye varias opciones para realizar backup de una base de datos.

Full Database Backup

Tiene como propósito capturar toda la data que se encuentra almacenada dentro de una base de datos. Asimismo, es posible utilizarlo para recrear una base de datos en su totalidad. Este método de respaldo se encuentra disponible sin importar el modelo de recuperación configurado para la base de datos.

Debido a que las copias de respaldo no son instantáneas y pueden ejecutarse, mientras existen usuarios conectados, existe la posibilidad de presentarse alguna inconsistencia. Sin embargo, SQL Server no permite que esto suceda forzando una serie de pasos durante el backup.

- Lock de la base de datos, bloqueando todas las transacciones.
- Coloca una marca en el log de transacciones.
- Retira el lock sobre la base de datos.
- Backup de la base de datos.
- Lock de la base de datos, bloqueando todas las transacciones.
- Coloca una marca en el log de transacciones.
- Retira el lock sobre la base de datos.
- Extrae todas las transacciones entre las dos marcas del log de transacciones y son agregadas al backup.

El siguiente es el comando básico para ejecutar un full backup.

```
USE MASTER  
GO  
  
BACKUP DATABASE ADVENTUREWORKS2014  
TO DISK='C:\BACKUP\ADVENTUREWORKS2014.BAK'  
WITH INIT  
GO
```

Aparece el resultado de la ejecución

Mensajes

Se han procesado 24272 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Data' en el archivo 1.
Se han procesado 2 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Log' en el archivo 1.
BACKUP DATABASE procesó correctamente 24274 páginas en 7.802 segundos (24.306 MB/s).

Este tipo de backup se debe utilizar en los siguientes casos:

- Cuando se quiere replicar el ambiente de producción para poder analizar el compartimiento de la base de datos o de la aplicación que utiliza la base de datos.
- Como mínimo toda base de datos debe tener un backup de este tipo. Debido que ante cualquier falla de la base de datos se puede volver a levantar la data hasta el instante en que se realizó el último backup full.

Backup Diferencial

Capturan, únicamente, la data que ha sido modificada desde el último backup full.

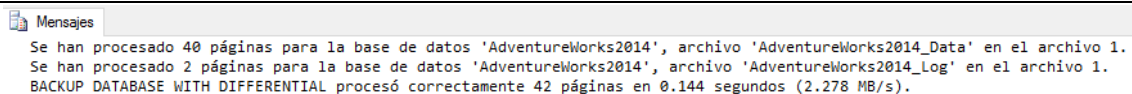
Se encuentra disponible sin importar el modelo de recuperación configurado para la base de datos. Cada backup diferencial contiene toda la información de los previos backups diferenciales, debido a que no es un backup incremental.

El siguiente es el comando para ejecutar un backup diferencial.

```
USE MASTER  
GO
```

```
BACKUP DATABASE ADVENTUREWORKS2014  
TO DISK='C:\BACKUP\ADVENTUREWORKS2014DIFF.BAK'  
WITH DIFFERENTIAL  
GO
```

Aparece el resultado de la ejecución:



Mensajes

Se han procesado 40 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Data' en el archivo 1.
Se han procesado 2 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Log' en el archivo 1.
BACKUP DATABASE WITH DIFFERENTIAL procesó correctamente 42 páginas en 0.144 segundos (2.278 MB/s).

Este tipo de backup se debe utilizar en los siguientes casos:

- Cuando se tienen aplicaciones con cierres contables diarios. Por ejemplo, aplicaciones que interactúan con una bolsa de valores que tienen un horario de cierre para la mañana y otro para la tarde. Aquí se podría realizar un backup diferencial después de cada cierre, para resguardar la data ingresada entre cada cierre. Por supuesto el primer backup tendría que ser de tipo Full.

Backup de Transaction Log

Este tipo de backups sólo puede ser ejecutado sobre bases de datos que tengan configurado el modelo de recuperación en Full o Bulk-Logged. Adicionalmente, sólo podrán ejecutarse después que se haya completado un full backup.

Los log backups contienen el active log. Además, SQL Server realiza un backup de todas las transacciones hasta que se encuentra con una transacción abierta.

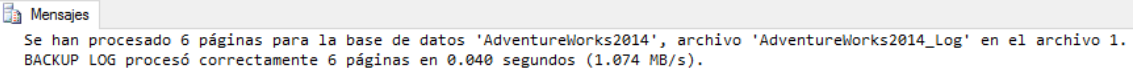
El siguiente es el comando básico para ejecutar un Transaction log backup.

```
USE MASTER  
GO
```

```
BACKUP LOG ADVENTUREWORKS2014  
TO DISK='C:\BACKUP\ADVENTUREWORKS2014.TRN'
```

```
WITH INIT  
GO
```

El resultado es:



Mensajes

Se han procesado 6 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Log' en el archivo 1. BACKUP LOG procesó correctamente 6 páginas en 0.040 segundos (1.074 MB/s).

Este tipo de backup se debe utilizar:

- Para aplicaciones consideradas críticas y altamente transaccionales, en las cuales no se tolera perder información.

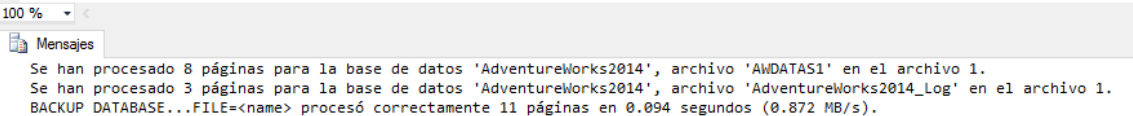
Filegroup Backup

Representa una alternativa para los full backups. Permite capturar la data de un filegroup individual en lugar de toda la base de datos. Se requiere que la base de datos se encuentre configurada con el recovery model Full o Bulk-Logged.

El siguiente es el comando básico para ejecutar un backup diferencial.

```
USE MASTER  
GO  
  
BACKUP DATABASE ADVENTUREWORKS2014  
FILEGROUP='SECUNDARIO'  
TO DISK='C:\BACKUP\ADVENTUREWORKS2014FGS.BAK'  
GO
```

El resultado será:



100 %

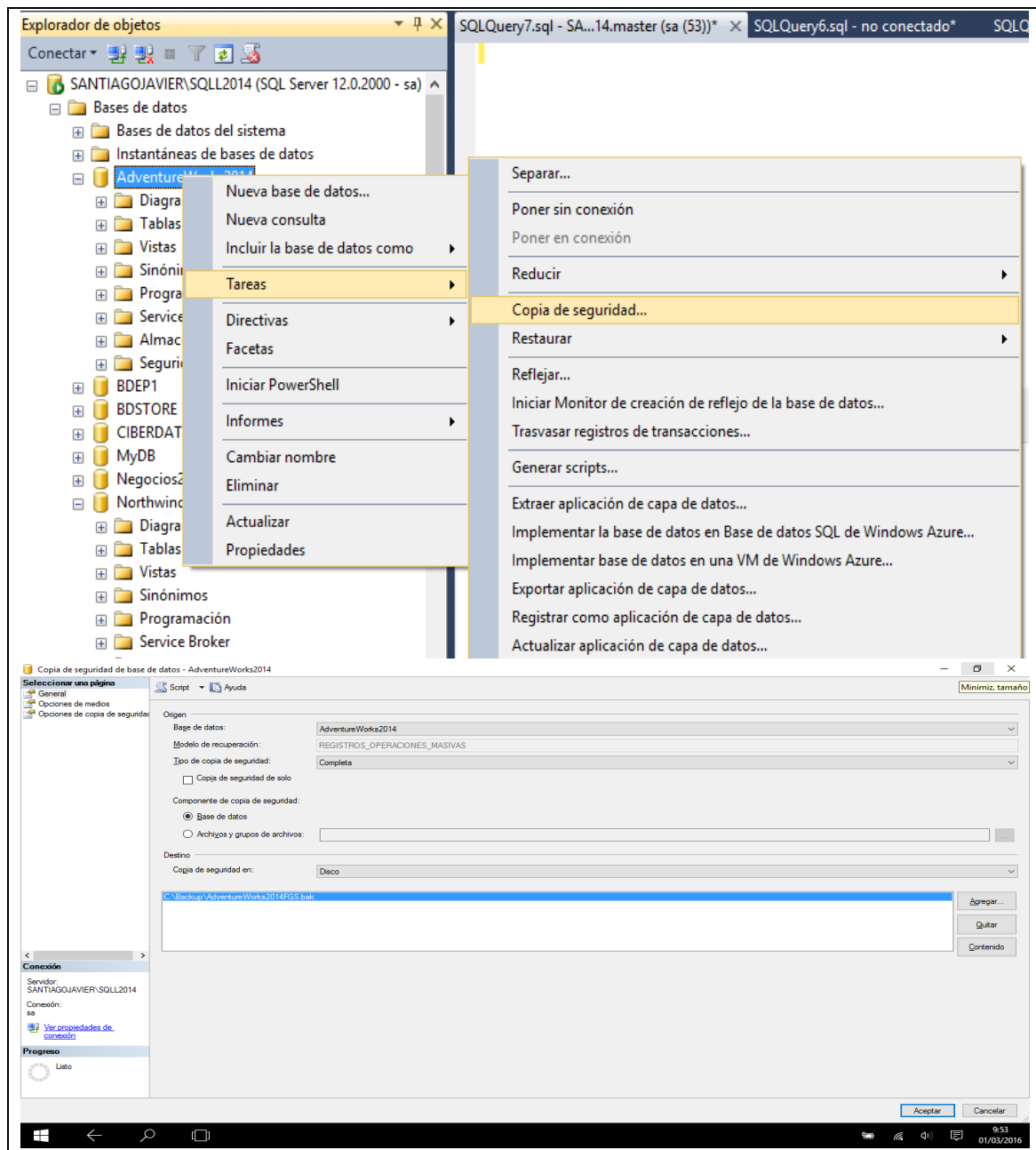
Mensajes

Se han procesado 8 páginas para la base de datos 'AdventureWorks2014', archivo 'AWDATAS1' en el archivo 1.
Se han procesado 3 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Log' en el archivo 1.
BACKUP DATABASE...FILE=<name> procesó correctamente 11 páginas en 0.094 segundos (0.872 MB/s).

Este tipo de backup se debe utilizar en los siguientes casos:

- Para bases de datos que tienen filegroup con data de sólo lectura y filegroups transaccionales, realizando el backup de la filegroup donde sólo se modifica la data, es decir, del filegroup transaccional.
- Cuando el tamaño de una base de datos representa una dificultad para ejecutar un backup o restore de toda la data, es decir, que el backup es demasiado grande y no se obtenga el espacio suficiente para realizar un full backup.

Desde el explorador de objetos también puede crear los backup.



Es importante mencionar que los archivos de backups no deben ser guardados en el mismo servidor, ni creados en el disco donde se encuentra instalado el sistema operativo:

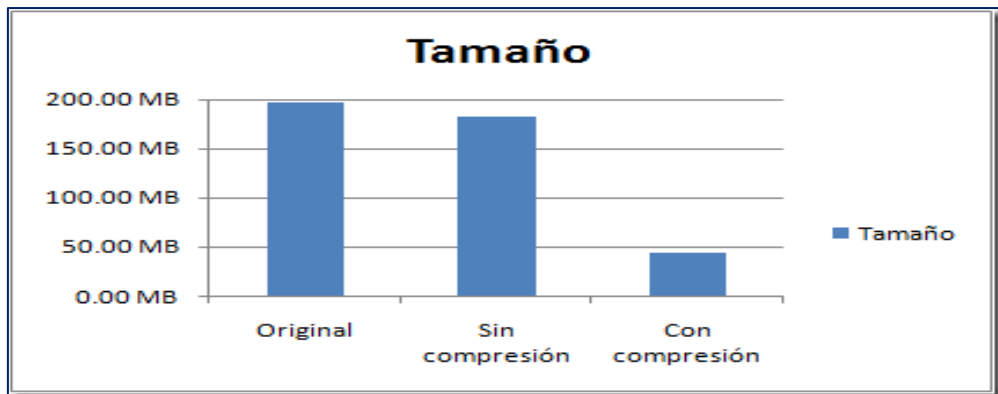
- Si los backups son creados en el mismo disco donde se encuentra el sistema operativo, puede ocasionar que éste se quede sin espacio libre, originando lentitud en el servidor y hasta puede originar que el servidor se reinicie. Es recomendable tener una unidad de disco exclusiva para backups.
- Si los archivos de backups se mantienen en el mismo servidor donde se encuentran las bases de datos, se corre el riesgo que, si este servidor tuviera algún problema de disco, los backups también se perderían. Es recomendable mover los archivos de backup a otra ubicación física o en todo caso, al sistema de cinta.

Backup Compression

Esta nueva funcionalidad sólo está disponible en las versiones Enterprise y Developer a partir de SQL Server 2012.

Ventajas y desventajas del Backup Compression

- Ventaja: al comprimir un respaldo, éste tomará menos tiempo en ejecutarse y menos accesos a los discos (IO).
- Desventaja: alto costo en CPU, por lo que se recomienda tener cuidado al generar el respaldo, ya que podría afectar a otras operaciones que se estén realizando en el mismo equipo durante la ejecución.



La primera alternativa es establecer que todos los backups que se hagan, en la instancia especificada, sean con compresión. Esto se hace estableciendo un parámetro de la configuración de la instancia, ya sea con instrucciones T-SQL o a través de la consola de administración SQL Server Management Studio.

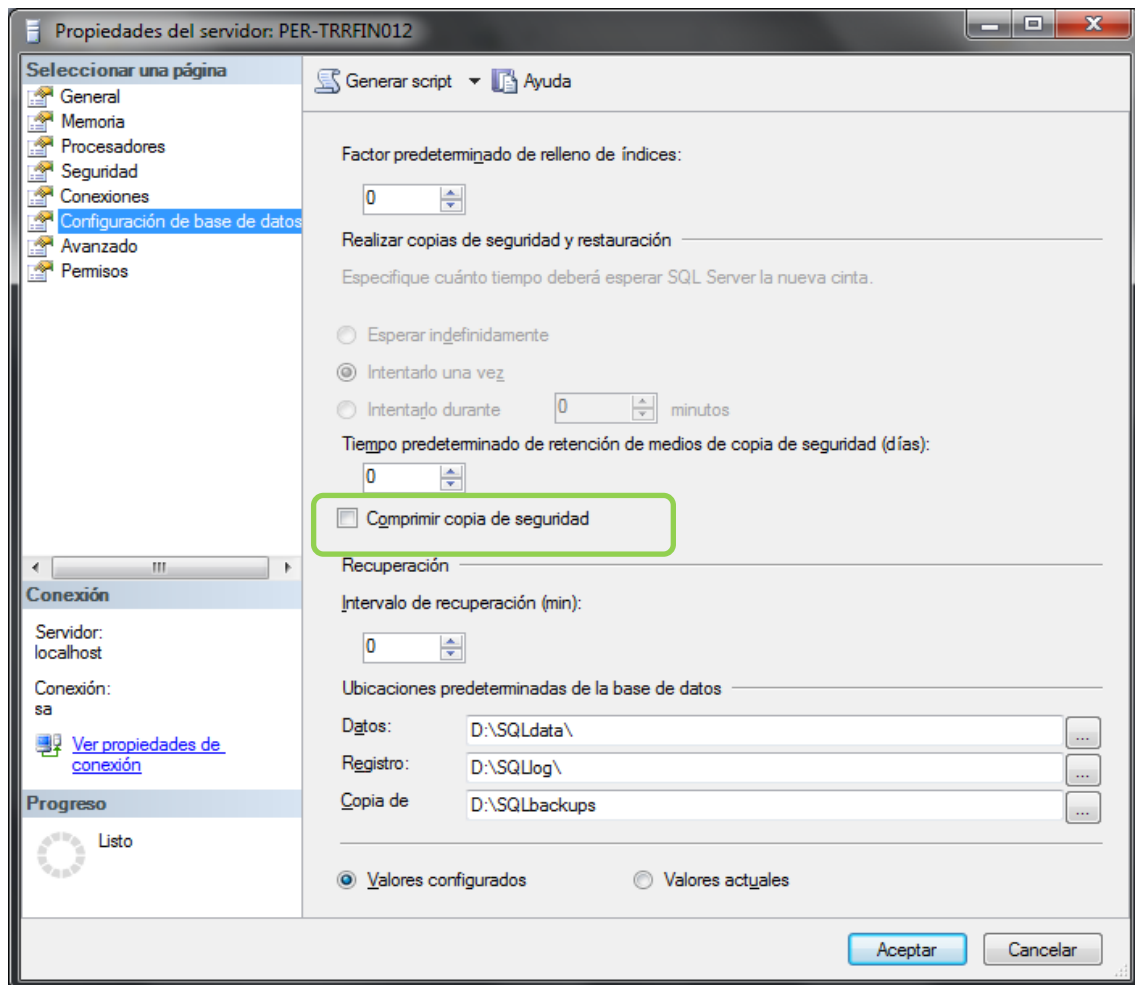
Si se realiza con T-SQL las instrucciones serían algo como lo que se muestra a continuación.

```
USE MASTER;  
GO  
  
EXEC sys.sp_configure 'backup compression default', '1';  
GO  
  
RECONFIGURE WITH OVERRIDE;  
GO
```

El resultado es el cambio de la activación del backup comprimido.

Mensajes
Se ha cambiado la opción de configuración 'backup compression default' de 1 a 0. Ejecute la instrucción RECONFIGURE para instalar.

Si se realiza a través del SQL Server Management Studio, se tendría que seleccionar el servidor en el Object Explorer, mostrar las Propiedades y en DatabaseSettings marcar CompressBackup.



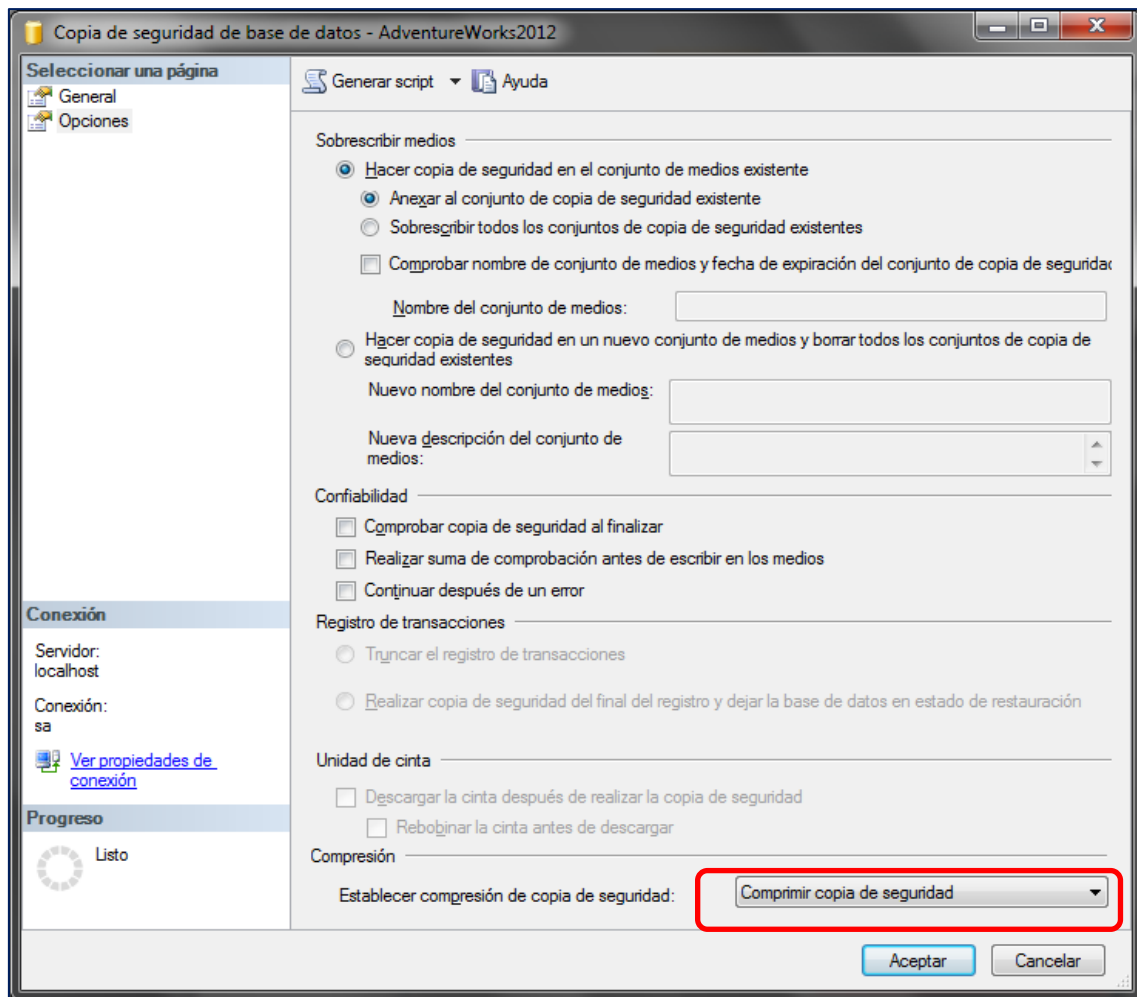
Cada respaldo que se haga cumplirá la característica de compresión o no compresión establecida. Sin embargo, si se quiere generar un respaldo con compresión o sin compresión, sin importar el valor que exista a nivel de configuración de la instancia, se puede realizar.

Si se hace con instrucciones es con la instrucción BACKUP DATABASE y usando WITH COMPRESSION para que tenga compresión o con WITH NO_COMPRESSION si no se desea que se comprima.

```
USE MASTER
GO
```

```
BACKUP DATABASE ADVENTUREWORKS2014
TO DISK='C:\BACKUP\20140228-1042_ADVENTUREWORKS2014.BAK'
WITH COMPRESSION,
FORMAT,
INIT,
NAME='ADVENTUREWORKS2014-FULL DATABASE BACKUP';
GO
```

Por otro lado, con SQL Server Management Studio se cambia el valor del comboboxSet backupcompression, a lo que se desea.



Técnicas para garantizar la integridad del backup

Como parte de la estrategia de respaldo de información, es importante considerar técnicas que permitan asegurar la integridad de los archivos de backup, así como, pruebas de restauración de los mismos. Un error en los archivos de backup puede ser fatal, ya que los respaldos son los que garantizan que, en caso de problemas en el sistema, se pueda recuperar la información. Asimismo, SQL Server incluye herramientas que permiten reducir el riesgo de que un archivo de backup se encuentre dañado al momento de necesitarlo para una recuperación de datos.

- Copias de respaldo espejadas (Mirror): esta opción solo está disponible en la edición Enterprise de SQL Server 2008 y permite en una sola sentencia, crear dos archivos de backup, la principal y el espejo. Ambas copias son exactamente iguales. Teniendo dos copias del mismo backup, se reduce la probabilidad de error. Otra ventaja es que una de las copias se puede almacenar en el disco local del servidor de base de datos mientras que la otra, se puede almacenar en un servidor remoto, a través de carpetas compartidas en la red.
- Checksum: realiza una suma de comprobación cuando lee una página de datos, cuyos cálculos se almacenan en el archivo de backup. Al momento de hacer la restauración, se puede usar la suma de comprobación para verificar si el archivo de backup está dañado.

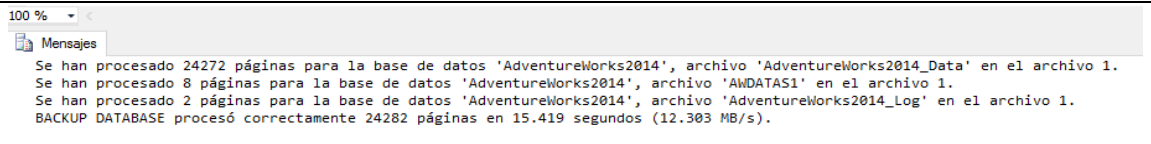
Cuando se utiliza compresión de backup, SQL Server por defecto, ejecuta operaciones de checksum.

USE MASTER

GO

```
BACKUP DATABASE ADVENTUREWORKS2014  
TO DISK='C:\BACKUP\AW12FULL.BAK'  
MIRROR TO DISK='C:\BACKUP_MIRROR\AW12FULL.BAK'  
WITH FORMAT, CHECKSUM  
GO
```

El resultado se verá así:



Una consideración importante para la ejecución de esta sentencia es que, para poder grabar backups en carpetas compartidas en la red, es necesario que la cuenta del servicio de SQL Server tenga permisos de escritura sobre dicha carpeta. Si se trabaja con cuentas del sistema, como por ejemplo, Local System, solo se podrá grabar backups en carpetas locales.

Tanto el backup con espejo como el checksum recargan el trabajo del backup, por lo que puede haber mayor uso de CPU y mayor demora en la ejecución. Pero esto es algo aceptable considerando los beneficios.

6.1.2 Restaurando una copia de seguridad.

Estrategia de prueba de recuperación

Es importante asegurarse que las copias de respaldo se están llevando a cabo correctamente. Por eso es aconsejable implementar estrategias de prueba y validación de los backup.

Una forma de validar que las copias de seguridad se encuentran bien es usando el comando RESTORE VERIFYONLY. Este comando lee el archivo de backup de la misma forma que lo haría en un proceso de restauración, pero no llega a restaurar la base de datos. Si la verificación del archivo de backup es correcta, se da por válida la copia de seguridad. Pero si se obtiene un error, ese backup no puede ser considerado para operaciones de recuperación.

```
RESTORE VERIFYONLY  
FROM DISK='C:\BACKUP\AW12FULL.BAK';  
GO
```

Una técnica muy completa para verificar la integridad de los backups es simular la restauración de la base de datos en una ubicación alternativa. El objetivo de esta técnica es verificar la integridad, no solo de los archivos de backup, si no de la base de datos misma luego de restaurada. En resumen, los pasos son los siguientes:

1. Verificar cuáles son los archivos contenidos en el backup.
2. Restaurar la base de datos de prueba en una ubicación alternativa, en estado de recuperación.

3. Recuperar la base de datos de prueba, pero con acceso restringido.
4. Ejecutar una verificación de la base de datos de prueba.
5. Eliminar la base de datos de prueba.

PASO 1: Este paso es necesario para ejecutar correctamente el paso 2. Restaurar la base de datos en una ubicación alternativa por lo general implica mover los archivos físicos, ya que no siempre contamos con un servidor para pruebas que tenga la misma estructura de almacenamiento (storage) que el servidor de producción. Con el comando RESTORE FILELISTONLY obtenemos la lista de archivos físicos contenidos en el backup cuya integridad queremos verificar:

```
RESTORE FILELISTONLY
FROM DISK='C:\BACKUP\AW12FULL.BAK';
GO
```

El resultado será:

	LogicalName	PhysicalName	Type	FileGroupName	Size	MaxSize	FileId	Create
1	AdventureWorks2014_Data	C:\Program Files (x86)\Microsoft SQL Server\MSAS...	D	PRIMARY	215220224	35184372080640	1	0
2	AWDATAS1	C:\DATA\AWDATAS1.NDF	D	SECUNDARIO	10485760	104857600	3	4500C
3	AdventureWorks2014_Log	C:\Program Files (x86)\Microsoft SQL Server\MSAS...	L	NULL	2097152	2199023255552	2	0

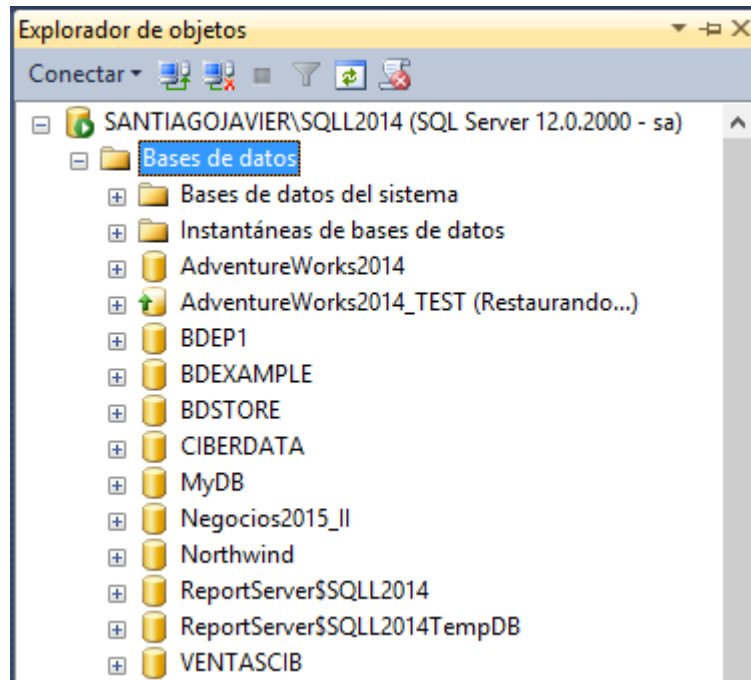
PASO 2: Procedemos a restaurar la base de datos desde el archivo de backup que utilizamos en el paso anterior, utilizando un nombre diferente para resaltar que es una base de datos de prueba. Con la información de los archivos físicos obtenida en el paso anterior y con el argumento MOVE..TO le indicamos a SQL Server donde debe copiar dichos archivos. Nótese que al final de la sentencia se incluye el argumento NORECOVERY, con el objetivo de dejar la base de datos en estado de recuperación para evitar que alguien pueda conectarse.

```
RESTORE DATABASE ADVENTUREWORKS2014_TEST
FROM DISK='C:\BACKUP\AW12FULL.BAK'
WITH
MOVE 'ADVENTUREWORKS2014_DATA'
    TO 'C:\TEST_DATA\ADVENTUREWORKS2014_DATA_TEST.TMDF',
MOVE 'AWDATAS1'
    TO 'C:\TEST_DATA\AWDATAS1_TEST.TNDF',
MOVE 'ADVENTUREWORKS2014_LOG'
    TO 'C:\TEST_DATA\ADVENTUREWORKS2014_LOG_TEST.TLDF',
NORECOVERY
GO
```

El resultado se ve así:

Mensajes
Se han procesado 24272 páginas para la base de datos 'AdventureWorks2014_TEST', archivo 'AdventureWorks2014_Data' en el archivo 1.
Se han procesado 8 páginas para la base de datos 'AdventureWorks2014_TEST', archivo 'AWDATAS1' en el archivo 1.
Se han procesado 2 páginas para la base de datos 'AdventureWorks2014_TEST', archivo 'AdventureWorks2014_Log' en el archivo 1.
RESTORE DATABASE procesó correctamente 24282 páginas en 8.377 segundos (22.645 MB/s).

Si observamos el Explorador de Objetos de Management Studio, la base de datos restaurada no esta disponible para uso, si no que permanece en estado de recuperación. De esa forma ningún usuario puede conectarse:



PASO 3: El siguiente paso es recuperar la base de datos, pero siempre para evitar que algún usuario se conecte, se recupera en el modo de usuario restringido (RESTRICTED_USER). En este modo, solo usuarios administradores o propietarios de la base de datos se pueden conectar.

```
RESTORE DATABASE ADVENTUREWORKS2014_TEST  
WITH RECOVERY, RESTRICTED_USER  
GO
```

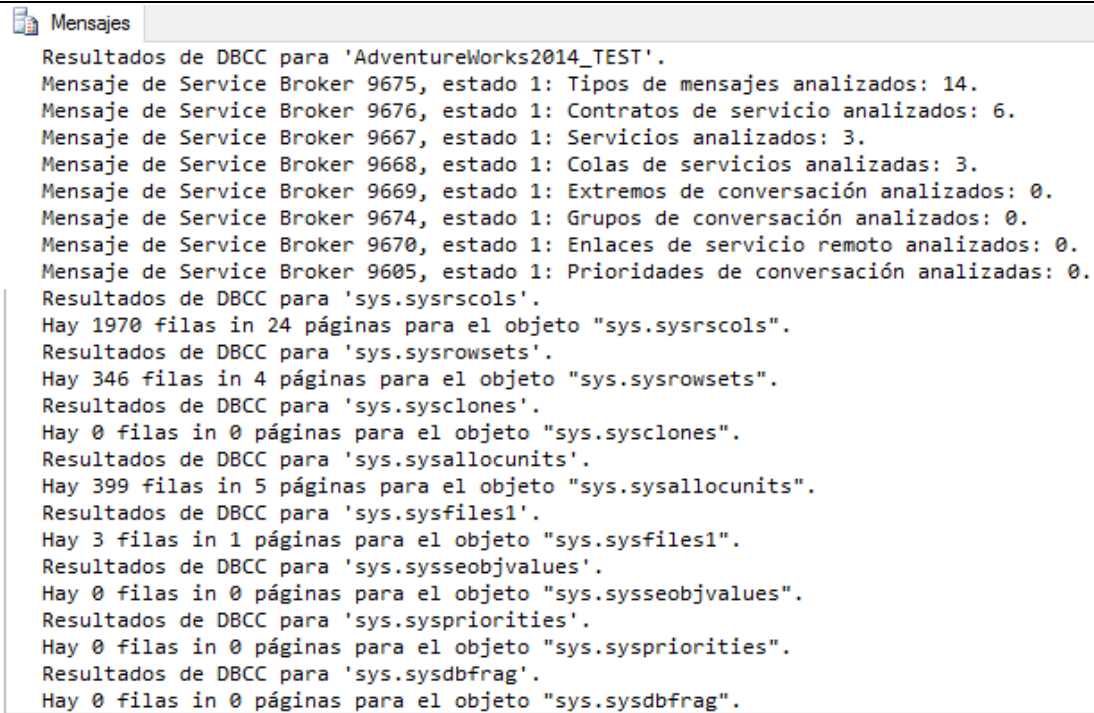
El resultado se verá así:

```
Mensajes  
RESTORE DATABASE procesó correctamente 0 páginas en 0.905 segundos (0.000 MB/s).
```

PASO 4: Una vez recuperada la base de datos de prueba, procedemos con la verificación utilizando el comando DBCC CHECKDB. El resultado de este comando debe indicar si la base de datos restaurada se encuentra bien o si presenta errores. Si sucede esto último, el backup utilizado no es válido, por lo que no se debe considerar en futuras acciones de recuperación.

```
DBCC CHECKDB (ADVENTUREWORKS2014_TEST)  
GO
```

A continuación, vemos una vista parcial del resultado.



```
Resultados de DBCC para 'AdventureWorks2014_TEST'.
Mensaje de Service Broker 9675, estado 1: Tipos de mensajes analizados: 14.
Mensaje de Service Broker 9676, estado 1: Contratos de servicio analizados: 6.
Mensaje de Service Broker 9667, estado 1: Servicios analizados: 3.
Mensaje de Service Broker 9668, estado 1: Colas de servicios analizadas: 3.
Mensaje de Service Broker 9669, estado 1: Extremos de conversación analizados: 0.
Mensaje de Service Broker 9674, estado 1: Grupos de conversación analizados: 0.
Mensaje de Service Broker 9670, estado 1: Enlaces de servicio remoto analizados: 0.
Mensaje de Service Broker 9605, estado 1: Prioridades de conversación analizadas: 0.
Resultados de DBCC para 'sys.sysrscsols'.
Hay 1970 filas in 24 páginas para el objeto "sys.sysrscsols".
Resultados de DBCC para 'sys.sysrowsets'.
Hay 346 filas in 4 páginas para el objeto "sys.sysrowsets".
Resultados de DBCC para 'sys.sysclones'.
Hay 0 filas in 0 páginas para el objeto "sys.sysclones".
Resultados de DBCC para 'sys.sysallocunits'.
Hay 399 filas in 5 páginas para el objeto "sys.sysallocunits".
Resultados de DBCC para 'sys.sysfiles1'.
Hay 3 filas in 1 páginas para el objeto "sys.sysfiles1".
Resultados de DBCC para 'sys.sysseobjvalues'.
Hay 0 filas in 0 páginas para el objeto "sys.sysseobjvalues".
Resultados de DBCC para 'sys.syspriorities'.
Hay 0 filas in 0 páginas para el objeto "sys.syspriorities".
Resultados de DBCC para 'sys.sysdbfrag'.
Hay 0 filas in 0 páginas para el objeto "sys.sysdbfrag".
```

PASO 5: Una vez terminada la prueba, eliminamos la base de datos creada:

```
DROP DATABASE AdventureWorks2014_TEST
GO
```

Los tipos de Restore se adecuan según el tipo de backup que se tiene, es decir, si se quiere realizar un restore de un archivo de backup de Log, se realiza un restore de Log.

Del mismo modo, para un backup full se realizará un restore full, para un backup diferencial se tendrá que realizar un restore diferencial.

Restore - Full Backup

Muchas operaciones de restore se inician recreando la base de datos en algún punto específico del tiempo y luego, se restauran subsiguientes backups para llevar la base de datos hacia un punto del tiempo más particular. Este proceso empieza con el restore de un full backup.

Mejores Prácticas: en caso la base de datos ya exista en la instancia, el restore de un full backup sobrescribe la base de datos con el mismo nombre. En caso la base de datos no exista, el proceso de restore crea los datafiles y filegroups antes de realizar el restore de la data.

Es recomendable no eliminar las bases de datos antes de un restore, si se tiene planeado sobrescribirla, esto debido a que la creación de los files puede consumir una significativa suma de tiempo.

El siguiente comando realiza un full database restore:

```
USE MASTER
GO
```

```
RESTORE DATABASE ADVENTUREWORKS2014
FROM DISK='C:\BACKUP\AW12FULL.BAK'
WITH REPLACE
GO
```

La opción REPLACE indica sobrescribir la base de datos existente, llamada AdventureWorks2014. Otras opciones importantes son:

- WITH RECOVERY: la base de datos se coloca on line y se aceptan transacciones.
- WITH NORECOVERY: la base de datos o filegroup permanece en estado RESTORING. Es posible restaurar backups adicionales, tal como diferencial o transaction log.

Restore–Backup Diferencial

Para restaurar un backup diferencial, primero se debe restaurar un backupfull con la opción NORECOVERY.

El siguiente comando realiza una restauración de backup diferencial, luego de restaurar el backup full.

```
--CREANDO BACKUP COMPLETO INICIAL
USE MASTER
GO

BACKUP DATABASE ADVENTUREWORKS2014
TO DISK = 'C:\BACKUP\AW.BAK'
WITH INIT
GO

--CREANDO BACKUP DIFERENCIAL
BACKUP DATABASE ADVENTUREWORKS2014
TO DISK = 'C:\BACKUP\AWDIFF.BAK'
WITH DIFFERENTIAL
GO

-- DESPUÉS DE ALGÚN TIEMPO
-- RESTAURANDO PRIMERO EL BACKUP COMPLETO SIN RECUPERAR
USE MASTER
GO

RESTORE DATABASE ADVENTUREWORKS2014
FROM DISK = 'C:\BACKUP\AW.BAK'
WITH NORECOVERY, REPLACE
GO

--RESTAURANDO LUEGO EL BACKUP DIFERENCIAL, RECUPERANDO
RESTORE DATABASE ADVENTUREWORKS2014
FROM DISK = 'C:\BACKUP\AWDIFF.BAK'
```

WITH RECOVERY
GO

Aparecerá el siguiente resultado:

Mensajes

Se han procesado 40 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Data' en el archivo 1.
Se han procesado 8 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Log' en el archivo 1.
Se han procesado 2 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Log' en el archivo 1.
RESTORE DATABASE procesó correctamente 50 páginas en 0.137 segundos (2.851 MB/s).

Restore – Transaction Log Backup:

Son utilizados para restaurar la base de datos a determinado punto del tiempo, usualmente, la última operación que fue ejecutada. Aplica solo para los modelos de recuperación Completo (Full) y de Registro Masivo (BulkLogged).

A continuación, se muestran dos ejemplos de restauración de base de datos usando backup de registro de transacciones.

El siguiente ejemplo utiliza un backup full, un backup diferencial y un backup del log de transacciones.

```
--CREANDO BACKUP FULL
BACKUP DATABASE ADVENTUREWORKS2014
TO DISK = 'C:\BACKUP\AWFULL.BAK'
WITH INIT
GO

--CREANDO BACKUP DIFERENCIAL
BACKUP DATABASE ADVENTUREWORKS2014
TO DISK = 'C:\BACKUP\AWDIFFE.BAK'
WITH DIFFERENTIAL
GO

--CREANDO BACKUP LOG
BACKUP LOG ADVENTUREWORKS2014
TO DISK = 'C:\BACKUP\AWLOG.TRN'
WITH INIT
GO

/***** RESTAURANDO *****/
--PRIMERO SE RESTAURA EL BACKUP FULL, SIN RECUPERAR
RESTORE DATABASE ADVENTUREWORKS2014
FROM DISK = 'C:\BACKUP\AWFULL.BAK'
WITH NORECOVERY, REPLACE
GO

--LUEGO SE RESTAURAR EL BACKUP DIFERENCIAL, SIN RECUPERAR
RESTORE DATABASE ADVENTUREWORKS2014
FROM DISK = 'C:\BACKUP\AWDIFFE.BAK'
WITH NORECOVERY
```


GO

```
--POR ÚLTIMO RESTAURAR EL BACKUP LOG, RECUPERANDO
RESTORE LOG ADVENTUREWORKS2014
FROM DISK = 'C:\BACKUP\AWLOG.TRN'
WITH RECOVERY
GO
```

Se mostrará el siguiente resultado:

Mensajes

Se han procesado 0 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Data' en el archivo 1.
 Se han procesado 0 páginas para la base de datos 'AdventureWorks2014', archivo 'AWDATAS1' en el archivo 1.
 Se han procesado 14 páginas para la base de datos 'AdventureWorks2014', archivo 'AdventureWorks2014_Log' en el archivo 1.
 RESTORE LOG procesó correctamente 14 páginas en 0.026 segundos (4.056 MB/s).

Dispositivo Lógico de Copia de Seguridad

Un dispositivo lógico es un nombre definido por el usuario que señala un dispositivo físico de copia de seguridad específico (un archivo de disco o unidad de cinta). La inicialización del dispositivo físico tiene lugar posteriormente, cuando se escribe una copia de seguridad en el dispositivo de copia de seguridad.

Para crear un dispositivo lógico de backup, se utiliza el procedimiento SP_addumpdevice (Transact-SQL).

Sintaxis:

```
sp_addumpdevice [ @devtype = ] 'device_type'
, [ @logicalname = ] 'logical_name'
, [ @physicalname = ] 'physical_name'
[ , { [ @cntrltype = ] controller_type |
      [ @devstatus = ] 'device_status' }
]
```

Argumentos:

[@devtype=] 'device_type'

Es el tipo de dispositivo de copia de seguridad. device_type es de tipo varchar (20), no tiene ningún valor predeterminado y puede tener uno de los valores siguientes.

Valor	Descripción
disk	Archivo de disco duro que se utiliza como dispositivo de copia de seguridad.
tape	Todos los dispositivos de cinta admitidos por Microsoft Windows.

[@logicalname =] 'logical_name'

Es el nombre lógico del dispositivo de copia de seguridad que se utiliza en las instrucciones BACKUP y RESTORE. logical_name es de tipo sysname, no tiene ningún valor predeterminado y no puede ser NULL.

[@physicalname =] 'physical_name'

Es el nombre físico del dispositivo de copia de seguridad. Los nombres físicos tienen que cumplir las reglas de nombres de archivo del sistema operativo o las convenciones de nomenclatura universal para los dispositivos de red, y deben incluir la ruta de acceso completa. physical_name es de tipo nvarchar(260), no tiene ningún valor predeterminado y no puede ser NULL.

Cuando cree un dispositivo de copia de seguridad en una ubicación de red remota, asegúrese de que el nombre con el que se haya iniciado el Motor de base de datos tenga permiso de escritura en el equipo remoto.

Si agrega un dispositivo de cinta, este parámetro tiene que ser el nombre físico asignado al dispositivo de cinta local por Windows; por ejemplo, \\.\TAPE0 para el primer dispositivo de cinta del equipo. El dispositivo de cinta tiene que estar en el equipo servidor; no se puede utilizar de forma remota. Incluya entre comillas los nombres que contengan caracteres no alfanuméricos.

[@cntrltype =] 'controller_type'

Obsoleto. Si se especifica, este parámetro se omite. Solo se admite para mantener la compatibilidad con versiones anteriores. En los nuevos usos de sp_addumpdevice se debe omitir este parámetro.

[@devstatus =] 'device_status'

Obsoleto. Si se especifica, este parámetro se omite. Solo se admite para mantener la compatibilidad con versiones anteriores. En los nuevos usos de sp_addumpdevice se debe omitir este parámetro.

En el ejemplo siguiente se muestra cómo agregar un dispositivo de copia de seguridad de disco llamado mydiskdump, con el nombre físico c:\Backup\dump.bak.

```
USE MASTER;  
GO  
  
EXEC SP_ADDUMPDEVICE 'DISK',  
                     'MYDISKDUMP',  
                     'C:\BACKUP\DUMP.BAK';  
GO
```

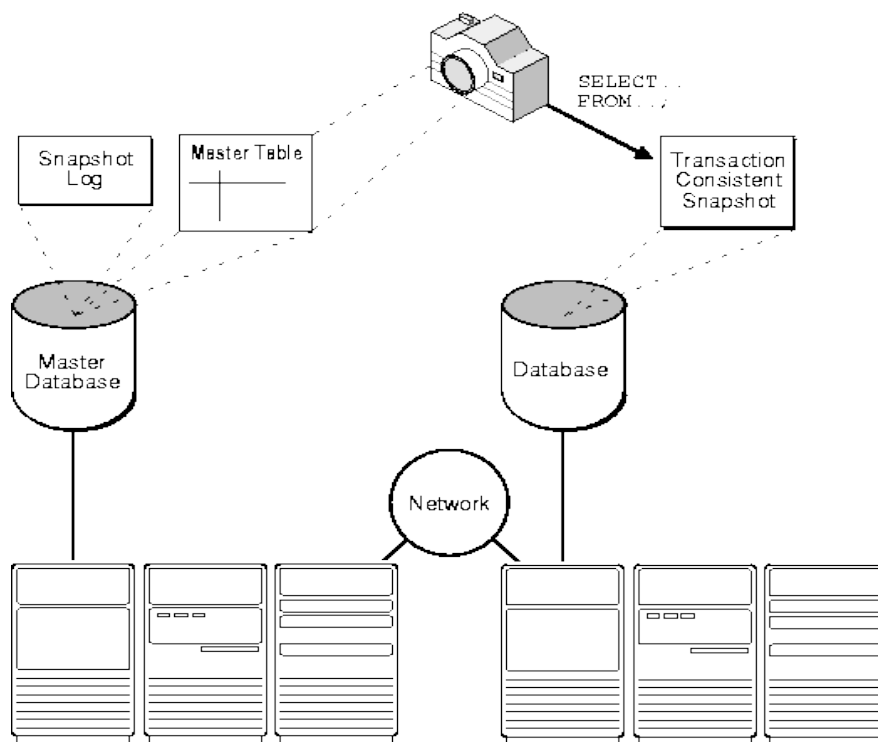
Ahora, utilizamos el dispositivo.

```
BACKUP DATABASE ADVENTUREWORKS2014  
TO MYDISKDUMP  
WITH INIT;  
GO
```

6.1.3 Base de datos Snapshot.

Un snapshot de base de datos es una copia de solo lectura de base de datos hecha en un momento en particular. Los usuarios pueden consultar el snapshot de la base de datos en la misma manera en la que consultarían la base de datos. Un snapshot de la base de datos puede ser usada para restaurar los datos y agregarla devuelta en la base de datos original rápida y fácilmente.

Los snapshots de la base de datos proveen la capacidad para los administradores de generar y usar una base de datos de solo lectura y vista estable. Una snapshot de la base de datos puede ser usada para recuperar datos perdidos por un cambio accidental hecho a la base de datos.



6.1.3.1 Creando una base de datos SnapShot.

La única manera de crear una instantánea de base de datos de SQL Server es utilizar Transact-SQL. SQL Server Management Studio no es compatible con la creación de instantáneas de bases de datos.

Se debe tener en cuenta, que el nombre lógico, debe ser igual al nombre lógico del datafile de la base de datos origen, y debe tener igual cantidad de los archivos.

Sintaxis:

```
CREATE DATABASE database_snapshot_name
ON
(
    NAME = logical_file_name,
    FILENAME = 'os_file_name'
) [ ,...n ]
```

AS SNAPSHOT OF source_database_name

[:]

En este ejemplo se crea una instantánea de la base de datos VentasCib. El nombre de la instantánea, VentasCib_dbss_1800, y el nombre del archivo, VentasCib_data_1800.ss, indican el tiempo de creación.

Paso 1: Se conoce el nombre lógico de los datafiles de la base de datos origen y su status que debe ser ONLINE.

USE MASTER

GO

EXEC SP_HELPDB VENTASCIB

GO

A continuación muestra la información de la base de datos:

Resultados Mensajes									
	name	db_size	owner	dbid	created	status			
1	VENTASCIB	5.23 MB	sa	14	Feb 25 2016	Status=ONLINE, Updateability=READ_WRITE, UserAcc...	compatibility_level		
							120		
	name	fileid	filename	filegroup	size	maxsize	growth	usage	
1	VENTASCIB	1	C:\Program Files (x86)\Microsoft SQL Server\MSAS...	PRIMARY	4288 KB	Unlimited	1024 KB	data only	
2	VENTASCIB_log	2	C:\Program Files (x86)\Microsoft SQL Server\MSAS...	NULL	1072 KB	2147483648 KB	10%	log only	

Paso 2: Se crear la BD SnapShot.

USE MASTER

GO

CREATE DATABASE VENTASCIB_DBSS1800

ON

(NAME = VENTASCIB,

FILENAME = 'C:\DATA\VENTASCIB_DATA_1800.SS')

AS SNAPSHOT OF VENTASCIB;

GO

6.1.3.2 Utilizando Snapshot para revertir cambios.

Si se dañan los datos de una base de datos en línea, revertir la base de datos a una instantánea de base de datos anterior puede ser, en algunos casos, una alternativa adecuada a restaurar la base de datos a partir de una copia de seguridad.

Por ejemplo, revertir una base de datos puede resultar útil para revertir un error grave del usuario que sea reciente, por ejemplo la eliminación de una tabla. Tenga en cuenta que se pierden todos los cambios realizados después de que se creara la instantánea.

En este ejemplo se considera que solo existe una instantánea en la base de datos VentasCib.

```
USE MASTER
GO

RESTORE DATABASE VENTASCIB
FROM DATABASE_SNAPSHOT = 'VENTASCIB_DBSS1800';
GO
```

Se revirtió la BD hasta el momento en que se creó el Snapshot.

Limitaciones y restricciones

Aunque el trabajar con snapshots ofrece una manera fácil y rápida de revertir todos los cambios aplicados en una base de datos hasta llevarla al punto de creación del snapshot, es cierto también que existen unas limitaciones y restricciones que hay que tener en cuenta por ejemplo:

- La creación de snapshot solo esta disponible en las versiones Enterprise a partir de SQL Server 2005.
- Sólo se permite una snapshot por cada base de datos.
- La base de datos debe estar online cuando se crea el snapshot.
- No se puede restaurar un backup sobre una base de datos que tiene un snapshot.
- No se puede eliminar una base de datos si esta tiene un snapshot creado.
- Si la base de datos esta corrupta entonces el snapshot también lo estará.

Cuando usar snapshots

Es recomendado usar snapshots en los siguientes casos

- Se quiere crear tener puntos de restauración antes de aplicar cambios importantes en la base de datos.
- En ambientes de pruebas cuando se requiere rápidamente volver a un estado previo luego de ejecutar varios procesos sobre los datos.
- Se quiere crear mantener temporalmente información histórica para la generación de reportes.
- Cuando se quiere utilizar una base de datos reflejada este disponible para el acceso a la generación de reportes mientras se realizan tareas de mantenimiento.

6.2 AUTOMATIZAR TAREAS EN SQL SERVER

Existen muchas tareas administrativas diarias que se deben realizar de manera rutinaria para administrar y mantener una base de datos. La automatización de dichas tareas minimiza la carga de trabajo administrativa, asociada a la administración de una base de datos. Además, permite identificar y resolver problemas antes de que éstos afecten a la operación de la base de datos.

El trabajo de un administrador conlleva deberes administrativos que no cambian de un día para otro y que pueden llegar a ser rutinarios. Es posible automatizar estas tareas rutinarias y configurar SQL Server para supervisar ciertos tipos de problemas antes de que aparezcan.

Beneficios de automatización

- Reducción de la carga de trabajo administrativa, con lo que se permite a los administradores de bases de datos, centrarse en otras funciones del trabajo, como planear cambios en la base de datos u optimizar su rendimiento, en lugar de estar pendientes de tareas de mantenimiento rutinarias.
- Reducción del riesgo de que se pasen por alto tareas de mantenimiento fundamentales.
- Reducción del riesgo de error humano.

Hay que tener en cuenta que la automatización implica delegación de tareas. Esto quiere decir que, aunque se le delegan al motor de base de datos la ejecución de ciertas tareas de manera automática, los administradores siguen siendo responsables de su ejecución, por lo tanto, deben incluir en sus actividades, el monitoreo de estas tareas para garantizar que se están ejecutando correctamente.

El agente de SQL Server

Es el componente de SQL Server, responsable de la automatización de las tareas administrativas de SQL Server.

Para que el Agente SQL Server ejecute trabajos, debe estar en funcionamiento siempre y debe contar con permisos suficientes. Al usar el Agente SQL Server, el administrador de las bases de datos, podrá programar tareas complejas y flexibles.

El Agente SQL Server se ejecuta como servicio de Windows. Dicho servicio tiene que estar en funcionamiento para poder ejecutar los trabajos programados. Las mejores prácticas sugieren configurar el Agente SQL Server para que se inicie automáticamente cada vez que Windows Server inicie.

Además, es posible configurar el servicio Agente SQL Server para que se reinicie automáticamente, si se detiene inesperadamente mediante el Administrador de configuración de SQL Server.

6.2.1 Creando tareas.

Un trabajo es una secuencia de tareas realizada secuencialmente por el Agente SQL Server. Un trabajo puede realizar principalmente tareas del siguiente tipo:

- Comandos de Transact-SQL.
- Aplicaciones de línea de comandos.
- Scripts ActiveX.

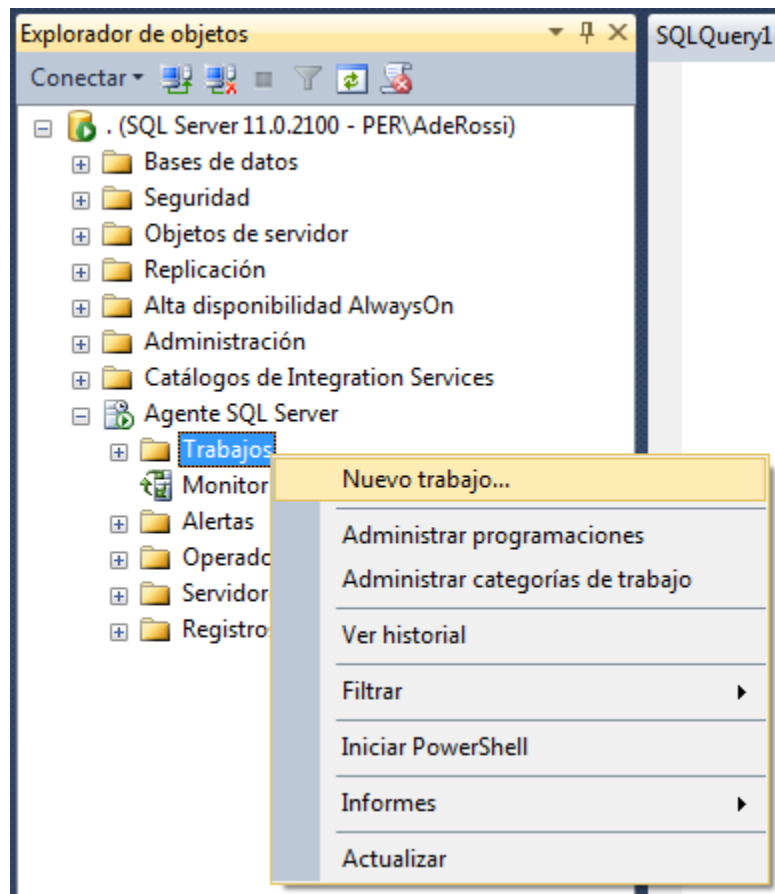
Además, puede ejecutar:

- Paquetes de Integration Services.
- Comandos y consultas de Analysis Services.
- Tareas de replicación.

Típicamente, estos últimos tipos de trabajos son creados automáticamente por SQL Server, como parte de la ejecución de otras herramientas.

La mejor manera de crear un trabajo es a través de Management Studio, pero también se puede ejecutar el procedimiento almacenado del sistema `sp_add_job` para crear un

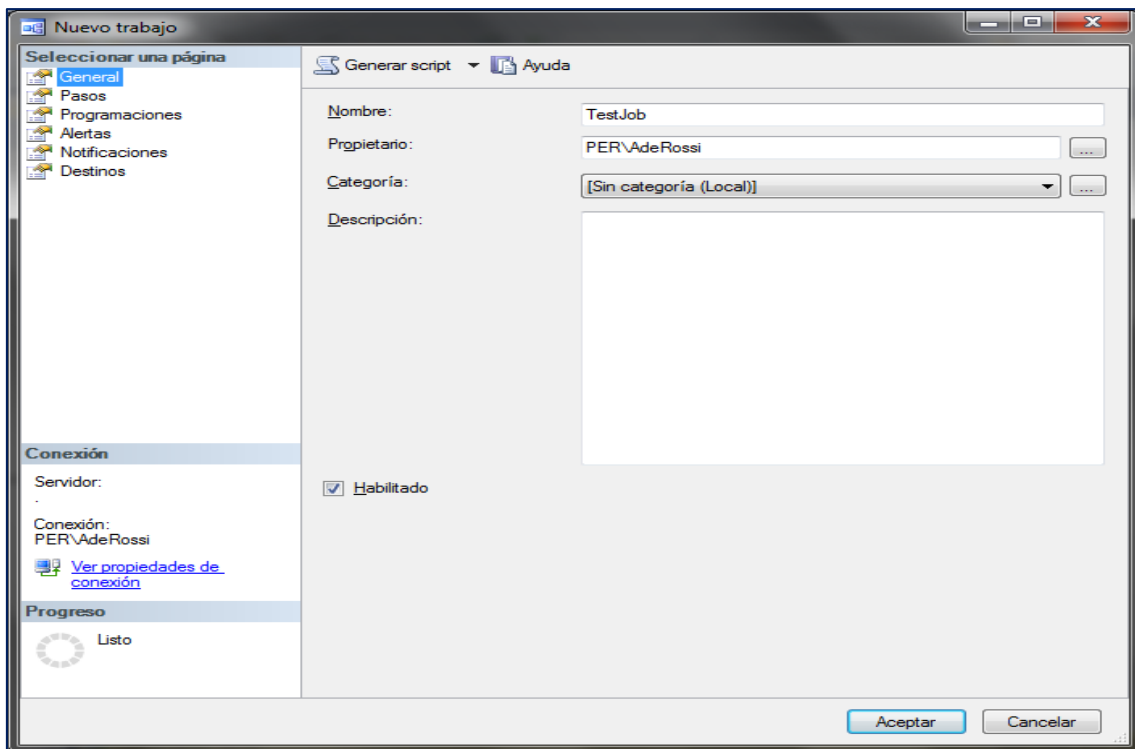
trabajo nuevo, aunque requiere el conocimiento completo de los parámetros requeridos.



Toda la información de los trabajos se guarda en la base de datos msdb. La tabla sysjobs tiene el detalle de los trabajos creados en la instancia.

Los trabajos se habilitan de manera predeterminada al momento de crearse. Pero si un trabajo se deshabilita, no podrá ejecutarse tal y como estaba programado. Sin embargo, un usuario todavía puede ejecutar manualmente un trabajo deshabilitado si lo inicia en Management Studio.

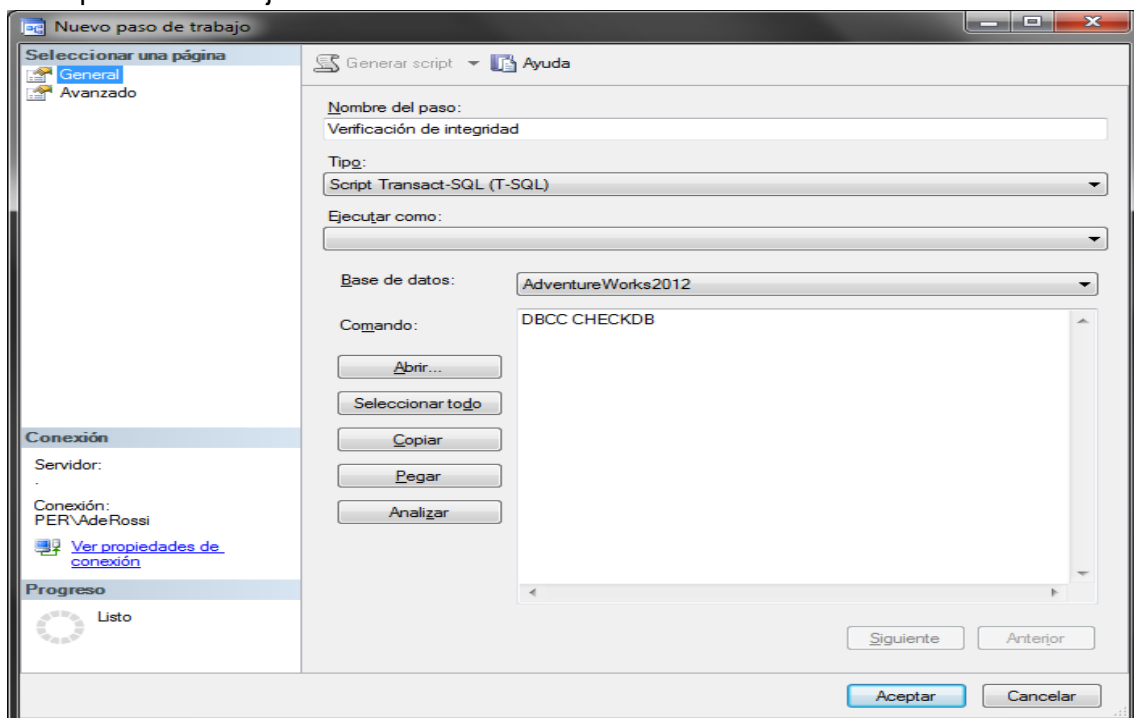
De forma predeterminada, el trabajo se va a ejecutar con los permisos de la cuenta del servicio del Agente SQL. Si requiere permisos diferentes se deben especificar en el trabajo.



Los trabajos están conformados por pasos. Los pasos son las tareas específicas que se van a ejecutar en el trabajo. Un trabajo puede tener uno o varios pasos.

Los pasos se crean en SQL Server Management Studio o también, a través del procedimiento almacenado del sistema `sp_add_jobstep`. Las definiciones de los pasos del trabajo se almacenan en la tabla del sistema `sysjobsteps`, en la base de datos `msdb`.

Los pasos pueden ejecutar sentencias Transact-SQL, comandos del sistema operativo, scripts ActiveX, pero sólo se puede especificar un tipo de ejecución para cada paso del trabajo.



Al crear los trabajos, un administrador de bases de datos debería especificar las medidas que SQL Server, debería tomar si un paso de trabajo se realiza correctamente o si se produce un error. De forma predeterminada, SQL Server avanza al paso de trabajo siguiente si se realiza correctamente y se detiene si se produce un error.

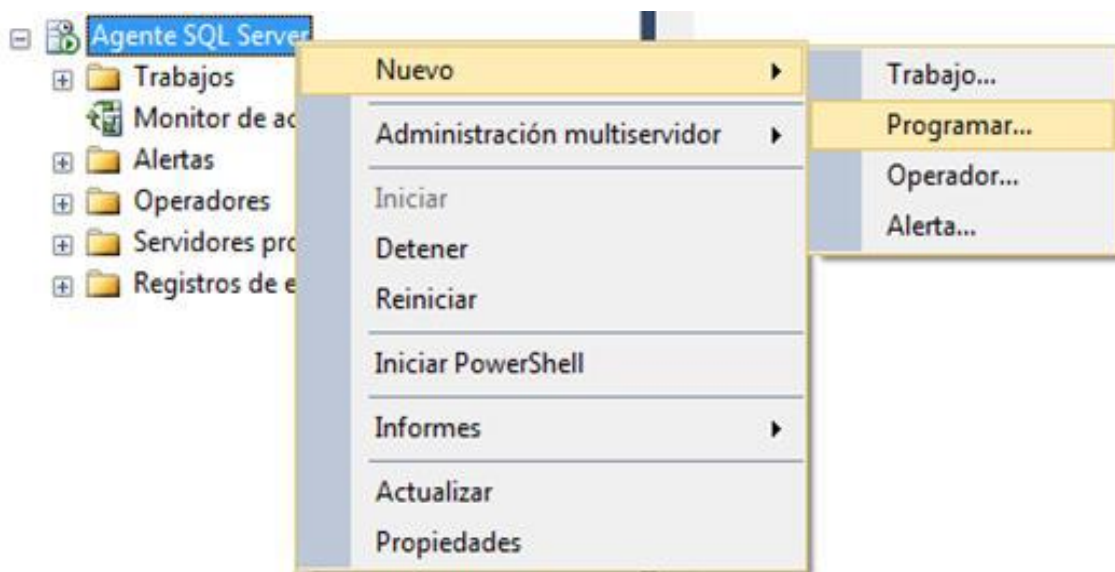
Sin embargo, los pasos de trabajo pueden ir a cualquier paso definido en el trabajo, tanto si es correcto como si se produce un error. El flujo lo puede definir el creador del trabajo.

Es posible indicar el número de veces que SQL Server debería intentar la ejecución de un paso de trabajo si se produce un error. También puede especificar el intervalo entre cada reintento (en minutos), de modo que se garantice su ejecución.

Creando programas de ejecución

Un programa de ejecución define la periodicidad con la que un trabajo se va a ejecutar. Cuando el trabajo es rutinario o repetitivo, se utilizan los programas para indicarle al agente de SQL Server, cuándo lo debe ejecutar.

Para crear una nueva programación se pueden seguir diferentes caminos. Una forma es a través del menú contextual del Agente SQL Server en el Explorador de Objetos de Management Studio. Al hacer clic derecho sobre el Agente de SQL Server aparece el menú Nuevo / Programar...



Otra forma es desde la opción de creación de un nuevo trabajo. La ventana de diálogo Nuevo Trabajo contiene la página Programaciones en la cual se debe hacer clic en el botón Nueva... para crear la programación. También se pueden seleccionar programaciones existentes.

En ambos casos se muestra la ventana de dialogo Nueva programación de Trabajo, la cual se explica a continuación.

Nueva programación de trabajo

Nombre: Trabajos en programación

Tipo de programación: Periódica ☒ **Habilitado**

Única repetición

Fecha: 13/11/2012 Hora: 05:13:09 p.m.

Frecuencia

Sucede: Semanal

Se repite cada: 1 semanas, el

☐ Lunes ☐ Miércoles ☐ Viernes ☐ Sábado
☐ Martes ☐ Jueves ☒ Domingo

Frecuencia diaria

☒ Sucede una vez a la(s): 12:00:00 a.m.
☐ Sucede cada: 1 horas A partir de: 12:00:00 a.m.
Finaliza: 11:59:59 p.m.

Duración

Fecha de inicio: 13/11/2012 ☐ Fecha de finalización: 13/11/2012
☒ Sin fecha de finalización:

Resumen

Descripción: Sucede el domingo de cada semana a las 12:00:00 a.m.. Se utilizará la programación que

Aceptar Cancelar Ayuda

Tipo de programación

- Periódica. Se repite de acuerdo al programa.
- Una Vez. Se ejecuta en la fecha y hora indicada.

Única Repetición.- Esta sección de la ventana se activa, solo si se ha escogido el tipo de programación Una Vez. En esta sección se especifica la fecha y la hora en la que se ejecutará por única vez el trabajo.

Frecuencia

Esta sección de la ventana se activa, solo si se ha escogido el tipo de programación Periódica. Las frecuencias pueden ser:

Diaria: se especifica cada cuántos días se repite el trabajo. Si se indica el número 1, el trabajo se repetirá a diario:

Frecuencia

Sucede: Diaria

Se repite cada: 1 días

Semanal: se especifica cada cuántas semanas se repite el trabajo. Si se indica el número 1, el trabajo se repetirá todas las semanas. Adicionalmente, se indica qué días de la semana se va a ejecutar el trabajo:

The 'Frecuencia' window shows the 'Sucedee:' dropdown set to 'Semanal'. Below it, 'Se repite cada:' is set to '1' in a spinner box, followed by 'semanas, el'. There are two rows of checkboxes for days of the week: the first row has 'Lunes', 'Miércoles', 'Viernes', and 'Sábado'; the second row has 'Martes', 'Jueves', and 'Domingo'. The 'Domingo' checkbox is checked.

Mensual: se especifica en qué día y cada cuánto meses se repite el trabajo. Si se indica el número 1, el trabajo se repetirá todos los meses. Adicionalmente, se puede indicar un día de ejecución, en base a si es el primer día del mes o el último día del mes, entre otros:

The 'Frecuencia' window shows the 'Sucedee:' dropdown set to 'Mensual'. Below it, there are two radio buttons: 'El día' (unselected) and 'El' (selected). The 'El' option has two sub-options: '1' in a spinner box followed by 'de cada' and '1' in a spinner box followed by 'meses'; and 'último' in a dropdown box followed by 'día' in a dropdown box followed by 'de cada' and '1' in a spinner box followed by 'meses'.

Frecuencia Diaria: una vez establecida la frecuencia, se define si el trabajo se va a ejecutar una vez en el día o varias veces en el día:

The 'Frecuencia diaria' window shows two radio buttons: 'Sucedee una vez a la(s):' (unselected) and 'Sucedee cada:' (selected). The 'Sucedee cada:' option has a spinner box set to '3' followed by a dropdown box set to 'horas'. To the right, there are two time input fields: 'A partir de:' set to '12:00:00 a.m.' and 'Finaliza:' set to '11:59:59 p.m.'.

Duración: mediante la duración, se especifica el intervalo de tiempo en el que se ejecutará el trabajo. Se puede especificar una fecha de inicio y una fecha de fin. Pero por lo general, los trabajos periódicos se ejecutan sin fecha de finalización:

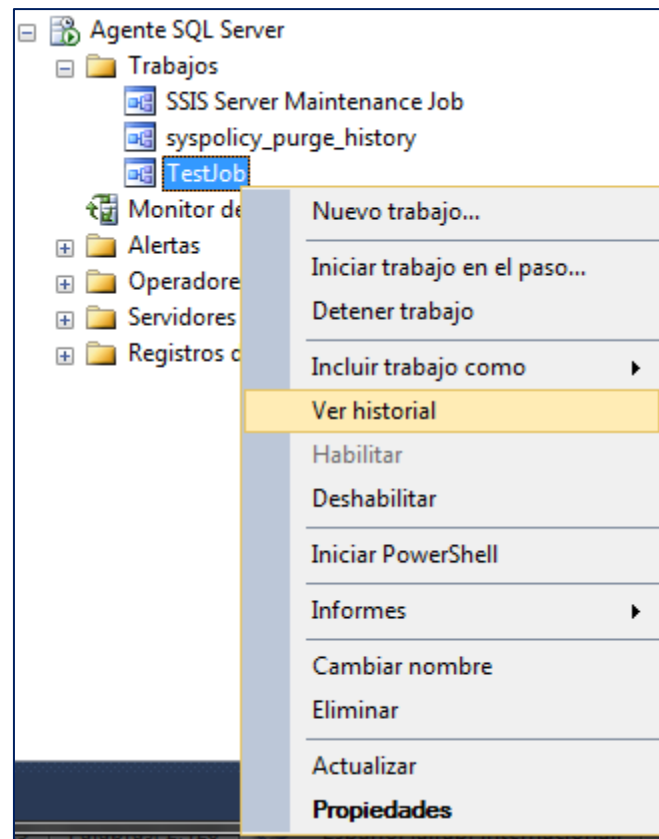
The 'Duración' window shows two date input fields: 'Fecha de inicio:' set to '10/10/2012' and 'Fecha de finalización:' set to '10/10/2012'. There are two radio buttons: 'Fecha de finalización:' (unselected) and 'Sin fecha de finalización:' (selected).

Resumen: el resumen explica en texto, la configuración definida para la programación. Una vez completada la información se hace clic en el botón Aceptar y la programación queda registrada.

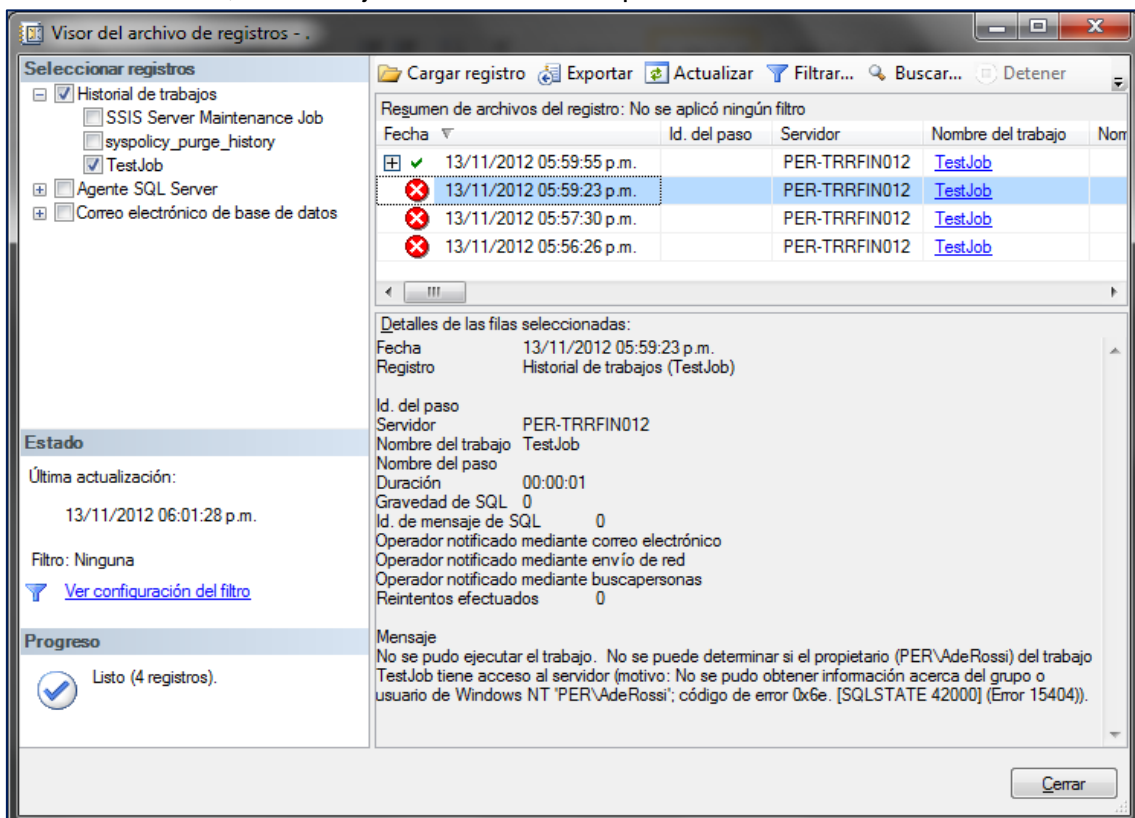
Ver la historia de un trabajo

Cada vez que un trabajo se ejecuta, se guarda la historia en la base de datos msdb, específicamente en la tabla sysjobhistory.

Para ver el historial de un trabajo se hace clic derecho sobre el trabajo que se quiere analizar y se utiliza la opción Ver historial:



Esta opción muestra el Visor del archivo de registros, a través del cual se puede revisar detalladamente, la historia de ejecución del trabajo. Desde esta ventana se pueden ver los mensajes resultantes de la ejecución, en especial, cuando el trabajo termina con error, con el objetivo de solucionar problemas.

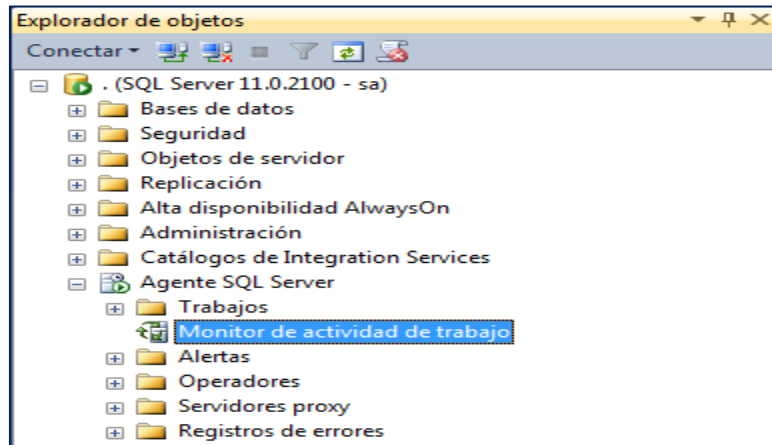


Otra manera de ver información de los trabajos y el historial es utilizando el monitor de actividad de trabajo del Agente SQL

El monitor de actividad de trabajo.

Es una herramienta de SQL Server Management Studio que le permite ver la tabla sysjobactivity.

Para mostrar el monitor de actividad, se hace doble clic en el nodo del Agente SQL, en el explorador de objetos.

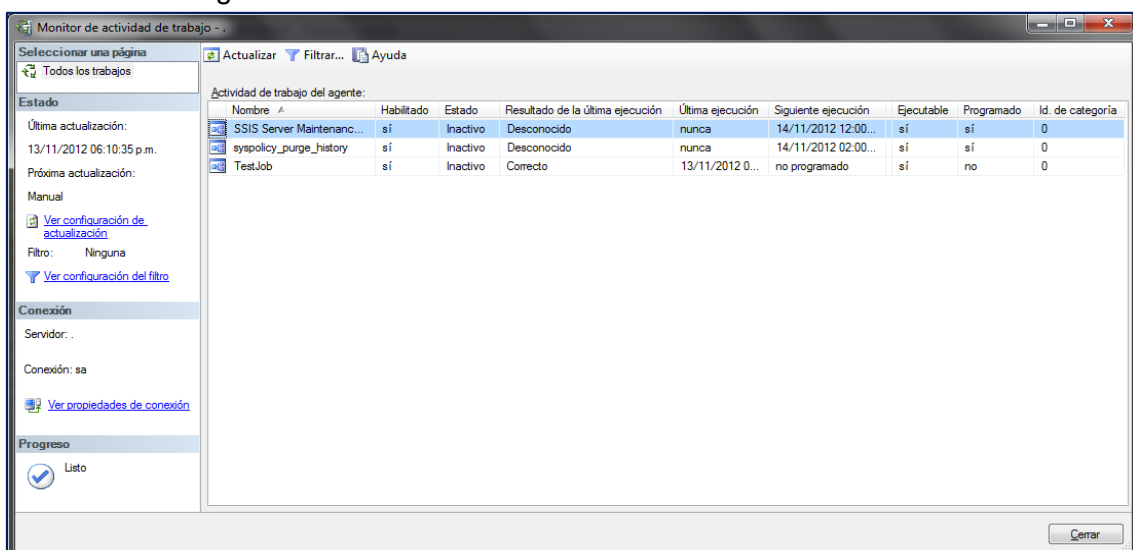


Con el monitor de actividad se puede visualizar cuáles son los trabajos programados para ejecutarse, el último resultado de los trabajos que se han ejecutado durante la sesión actual y cuáles se están ejecutando o están inactivos.

Si se produce un error en el servicio Agente SQL Server, se podrá determinar los trabajos que se estaban ejecutando examinando la sesión anterior en el Monitor de actividad de trabajo.

Desde el monitor de actividad se pueden realizar las siguientes tareas:

- Iniciar y detener trabajos.
- Ver las propiedades de los trabajos.
- Ver el historial de un trabajo concreto.
- Actualizar manualmente la información en la cuadrícula Actividad de trabajo del agente o establecer un intervalo de actualización automática haciendo clic en Ver configuración de actualización.



6.2.2 Creando alertas.

Las alertas nos brindan la capacidad de enviar notificaciones o realizar acciones basadas en ciertos eventos o condiciones propias de SQL Server o incluso de la máquina donde está corriendo la instancia de SQL Server.

Alertas de SQL Server Agent

Las alertas pueden ser de uno de los siguientes tres tipos:

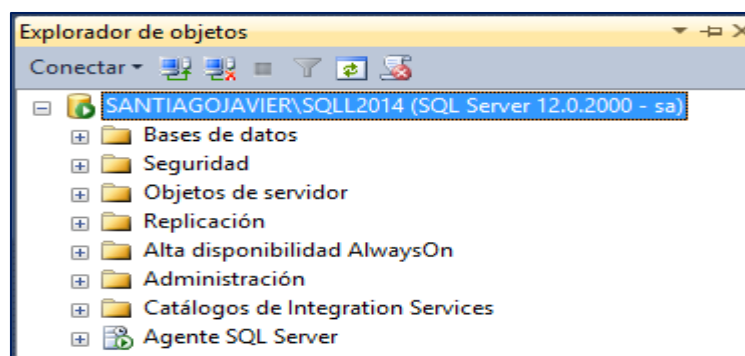
- Un evento de SQL Server
- Alerta de performance
- Un evento WMI (Windows Management Instrumentation)

Una alerta por evento es disparada por SQL Server basada ya sea en un código de error, o la severidad del mismo. Además podemos limitar las alertas a una db o a un texto específico devuelto en un mensaje de error. Cuando una alerta de SQL Server es creada, SQL Agent escanea los eventos de windows y busca alguna si alguna de las condiciones establecidas es válida.

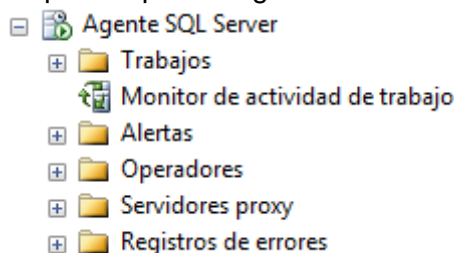
Alertas de performance se define teniendo en cuenta el "System monitor counter". Cuando una alerta es definida, debemos especificar un objeto, contador, e instancia que deseamos monitorear, estableciendo una condición del tipo igual mayor, mayor igual, menor, menor igual.... etc.

Para crear una alerta mediante un número de error (Utilizando SQL Server Management Studio)

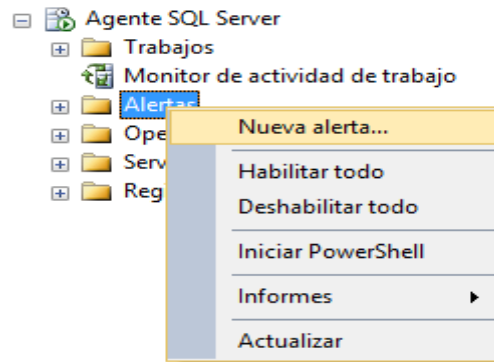
1. En el Explorador de objetos, haga clic en el signo más para expandir el servidor donde desea crear una alerta con un número de error.



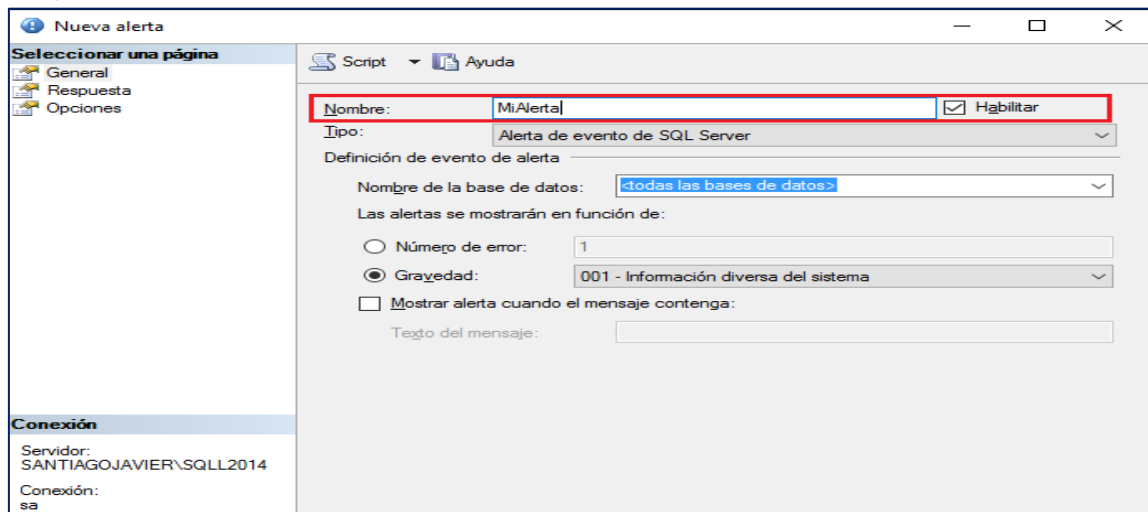
2. Haga clic en el signo más para expandir Agente SQL Server.



3. Haga clic con el botón secundario en Alertas y seleccione Nueva alerta.



4. En el cuadro de diálogo Nueva alerta, en el cuadro Nombre, escriba un nombre para esta alerta.



5. Active la casilla Habilitar para que la alerta se pueda ejecutar. De forma predeterminada, la opción Habilitar está activada.
6. En la lista Tipo, seleccione Alerta de evento de SQL Server.
7. En Definición de evento de alerta, en la lista Nombre de la base de datos, seleccione una base de datos para restringir la alerta a una base de datos específica.

8. En Las alertas se mostrarán en función de, haga clic en Número de error y escriba un número de error válido para la alerta. También puede hacer clic en Gravedad y seleccionar la gravedad específica que producirá la alerta.
9. Active la casilla correspondiente a Mostrar alerta cuando el mensaje contenga para restringir la alerta a una secuencia de caracteres en particular y, a continuación, escriba una palabra clave o una cadena de caracteres en el Texto del mensaje. El número máximo de caracteres es 100.

10. Haga clic en Aceptar.

Para crear una alerta mediante un número de error (Utilizando Transact SQL)

Se debe escribir el siguiente código.

USE MSDB

```
GO

EXEC MSDB.DBO.SP_ADD_ALERT
    @NAME='MIALERTA',
    @MESSAGE_ID=0,
    @SEVERITY=1,
    @ENABLED=1,
    @DATABASE_NAME='VENTASCIB',
    @EVENT_DESCRIPTION_KEYWORD='ATENCIÓN!! HA OCURRIDO UNA
OPERACIÓN EN LA BD VENTASCIB',
GO
```

Otro ejemplo de creación de alertas

```
USE MSDB;
GO

EXEC DBO.SP_ADD_ALERT
    @NAME = 'TESTALERT',
    @MESSAGE_ID = 0,
    @SEVERITY = 1,
    @NOTIFICATION_MESSAGE = 'LA BASE DE DATOS SERÁ RESPALDADO',
    @JOB_NAME = 'BACKUPAW';
GO
```

La alerta se producirá cuando realice backup a la base de datos AdventureWorks2014.

Esto no tiene sentido si no se notifica, el siguiente ejemplo, notifica a un correo electrónico de un operador definido previamente configurado.

```
USE MSDB;
GO

EXEC DBO.SP_ADD_NOTIFICATION
    @ALERT_NAME = 'TESTALERT',
    @OPERATOR_NAME = 'DBA',
    @NOTIFICATION_METHOD = 1;
GO
```

6.2.3 Creando planes de mantenimiento.

Los planes de mantenimiento permiten programar tareas de mantenimiento principales.

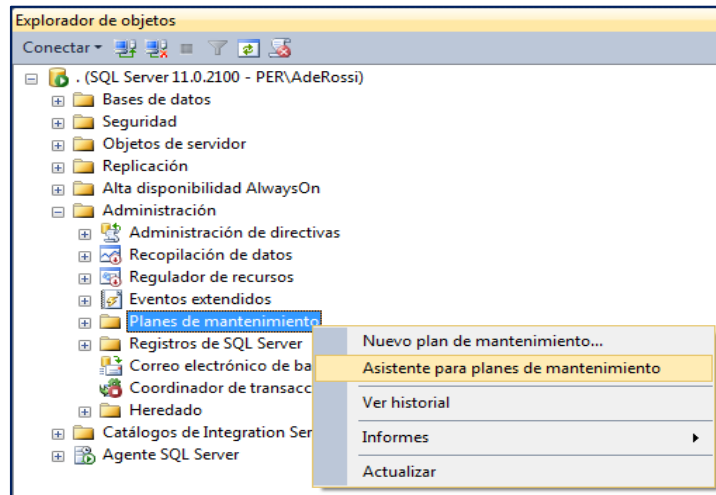
Se pueden programar backups de base de datos de usuario y del sistema, verificar el rendimiento de las bases de datos, realizar comprobaciones en busca de inconsistencias o corrupción de datos.

Los planes de mantenimiento crean trabajos del Agente SQL Server para que se lleven a cabo dichas tareas automáticamente, según el programa definido, aunque también se pueden ejecutar en cualquier momento a demanda del usuario.

Asistente de planes de mantenimiento

El asistente de planes de mantenimiento es una guía, paso a paso, para que el usuario pueda crear planes de mantenimiento de manera práctica y sencilla.

Para iniciar el asistente para planes de mantenimiento, se utiliza Management Studio. En el Explorador de objetos, en la carpeta Administración, se hace clic derecho en Planes de mantenimiento y se elige la opción Asistente para planes de mantenimiento.



Las tareas de mantenimiento que pueden programarse, para que se ejecuten automáticamente incluyen:

- Comprobar la integridad de la base de datos (CHECKDB). Realiza comprobaciones internas de coherencia de las páginas de datos y de índices de la base de datos.
- Reducir base de datos (SHRINK). Reduce el espacio en disco utilizado por la base de datos y los archivos de registro al quitar datos y páginas de registro vacíos.
- Reorganizar índice. Desfragmenta y compacta los índices clúster y no clúster de las tablas y vistas. Esto mejorará el rendimiento del examen de índice.
- Volver a generar índice. Reorganiza los datos existentes en las páginas de datos y de índices, al volver a generar los índices. Esto mejora el rendimiento de las búsquedas y los exámenes de índice. Esta tarea también optimiza la distribución de los datos y el espacio disponible en las páginas de índices, lo que permite un crecimiento futuro más rápido.
- Actualizar estadísticas. Garantiza que el optimizador de consultas tenga información actualizada sobre la distribución de los valores de datos en las tablas. Esto permite al optimizador, realizar evaluaciones más eficaces sobre las estrategias de acceso a datos.
- Limpieza de historial. Elimina los datos históricos de las operaciones de Copias de seguridad y restauración, Agente SQL Server y Plan de mantenimiento. Este asistente permite especificar el tipo y la antigüedad de los datos a eliminar.
- Ejecutar trabajo del Agente SQL Server. Permite seleccionar trabajos del Agente SQL Server para ejecutarlos como parte del plan de mantenimiento.
- Copia de seguridad de la base de datos (completa). Permite especificar las bases de datos de origen, los archivos o las cintas de destino, y las opciones de sobrescritura para una copia de seguridad completa (backup full).

- Copia de seguridad de la base de datos (diferencial). Permite especificar las bases de datos de origen, los archivos o las cintas de destino, y las opciones de sobrescritura para una copia de seguridad diferencial.
- Copia de seguridad de la base de datos (registro de transacciones). Permite especificar las bases de datos de origen, los archivos o las cintas de destino, y las opciones de sobrescritura para una copia de seguridad del registro de transacciones.
- Limpieza de mantenimiento. Quita archivos restantes de la ejecución de un plan de mantenimiento.

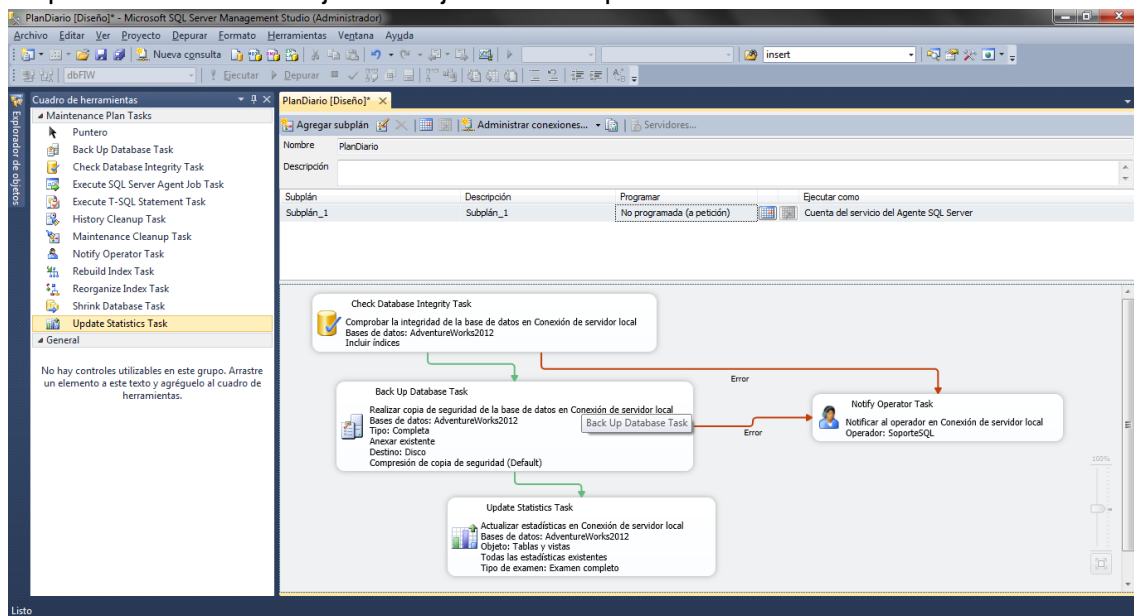
Opcionalmente, el resultado generado por las tareas de mantenimiento se puede grabar en una ruta, como un archivo de texto para que quede un registro. Esta información también se puede guardar en las tablas sysmaintplan_log y sysmaintplan_log_detail de la base de datos msdb.

Entorno de diseño de planes de mantenimiento

El entorno de diseño de planes de mantenimiento permite la creación de planes, así como, la modificación de planes de mantenimiento ya existentes, inclusive los creados con el asistente de planes de mantenimiento.

Asimismo, este entorno de diseño permite la creación o mejora de planes con algunas tareas adicionales que no se incluyen el asistente. Además de agregar funcionalidad al plan.

En el entorno de diseño se pueden agregar o eliminar tareas de mantenimiento, especificar la precedencia entre las tareas, definir una bifurcación de tareas o definir la ejecución de tareas en paralelo, así como, incluir tareas de notificación. En resumen, se puede controlar el flujo de la ejecución del plan.



Mediante el diseñador de planes de mantenimiento, se pueden crear sub planes, es decir, varios flujos independientes que son subconjuntos del plan de mantenimiento.

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.

<http://technet.microsoft.com/es-es/library/ms186858.aspx>

Aquí hallará los conceptos RESTORE TRANSACT SQL.

<http://technet.microsoft.com/es-es/library/ms186865.aspx>

En esta página, hallará los conceptos de BACK UP TRANSACT SQL.

<http://dblearner.com/como-garantizar-la-integridad-de-los-backup/>

En esta página, encontraras como garantizar la integridad de un backup.



MANEJO DE DATOS EN XML

LOGRO DE LA UNIDAD DE APRENDIZAJE

Al término de la unidad, el alumno genera consultas, actualizaciones y operaciones con datos en formato XML.

TEMARIO

7.1 Tema 15 : Base de datos relacionales para datos XML

- 7.1.1 : Introducción
- 7.1.2 : Tipos de datos XML
- 7.1.3 : For XML y OpenXML

7.2 Tema 16 : Procesamiento XML en SQL Server

- 7.2.1 : Almacenamiento de datos XML
- 7.2.2 : Recuperación de datos XML
 - 7.2.2.1 : Usar modo RAW
 - 7.2.2.2 : Usar modo AUTO
 - 7.2.2.3 : Usar modo EXPLICIT

ACTIVIDADES PROPUESTAS

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

7.1 BASE DE DATOS RELACIONALES PARA DATOS XML

7.1.1 Introducción.

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.

SQL Server proporciona una potente plataforma de desarrollo de aplicaciones para la gestión de datos semi-estructurada. Soporte para XML está integrada en todos los componentes de SQL Server e incluye lo siguiente:

- El tipo de datos XML. Los valores XML se pueden almacenar de forma nativa en una columna de tipo de datos XML que se pueden escribir de acuerdo con una colección de esquemas XML, o se deja sin tipo. Es posible indexar la columna XML.
- La posibilidad de especificar una consulta XQuery con datos XML almacenados en columnas y variables del tipo xml.
- Mejoras en OpenRowset para permitir la carga masiva de datos XML.
- La cláusula FOR XML, para recuperar datos relacionales en formato XML.
- La función OPENXML, para recuperar los datos XML en formato relacional.

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.

7.1.2 Tipo de dato XML.

Modelo de datos XML o Relacional

Si los datos están altamente estructurado con el esquema conocido, el modelo relacional es probable que funcione mejor para el almacenamiento de datos. SQL Server proporciona la funcionalidad requerida y herramientas que pueda necesitar. Por otro lado, si la estructura es semi-estructurada o no estructurada, o se desconoce, usted tiene que considerar la posibilidad de modelar estos datos.

XML es una buena opción si quieres un modelo independiente de la plataforma con el fin de asegurar la portabilidad de los datos mediante el uso de marcadores estructurales y semánticos. Además, es una opción adecuada si algunas de las siguientes propiedades son satisfechas:

- Sus datos son escasos o no sabes la estructura de los datos o la estructura de los datos puede cambiar significativamente en el futuro.
- Los datos representan árbol de contenidos, en lugar de referencias entre las entidades, y puede ser recursiva.
- El orden es inherente a sus datos.
- Que desea consultar a las partes de datos o actualización de la misma, en base a su estructura.

Si no se cumple ninguna de estas condiciones, se debe utilizar el modelo de datos relacional. Por ejemplo, si los datos están en formato XML, pero su aplicación sólo utiliza la base de datos para almacenar y recuperar los datos, una columna [n] varchar (max) es todo lo que necesita. El almacenamiento de los datos en una columna XML tiene beneficios adicionales. Esto incluye tener el motor a determinar que los datos son válidos o bien formado, y también incluye soporte para consultas y actualizaciones de grano fino en los datos XML.

Razones para almacenar datos XML en SQL Server

Las siguientes son algunas de las razones para utilizar las funciones XML nativos en SQL Server en lugar de la gestión de los datos XML en el sistema de archivos:

- Que desea compartir, consultar y modificar los datos XML de una manera eficiente y negociados, acceso a los datos de grano fino es importante para su aplicación. Por ejemplo, es posible que desee extraer algunas de las secciones dentro de un documento XML, o es posible que desee insertar una nueva sección sin tener que reemplazar todo el documento.
- Tiene datos relacionales y datos XML y desea interoperabilidad entre datos relacionales y XML dentro de su aplicación.
- Es necesario soporte de idiomas para la consulta y modificación de datos para aplicaciones de varios dominios.
- Desea que el servidor para garantizar que los datos está bien formada y opcionalmente también validar sus datos de acuerdo con los esquemas XML.
- ¿Quieres indexación de datos XML para el procesamiento eficaz de consulta y una buena escalabilidad, y el uso de un optimizador de consultas de primer nivel.
- ¿Quieres SOAP, ADO.NET y OLE DB de acceso a los datos XML.
- Desea utilizar la funcionalidad administrativa del servidor de base de datos para la gestión de los datos XML. Por ejemplo, esto sería la copia de seguridad, recuperación y replicación.

Si se cumple ninguna de estas condiciones, puede ser mejor para almacenar sus datos como un no-XML, tipo de objeto grande, como por ejemplo [n] varchar (max) o varbinary (max).

La siguiente consulta devuelve una lista de colecciones de esquemas XML de la base de datos AdventureWorks.

```
USE ADVENTUREWORKS2014
GO

SELECT *
FROM SYS.XML_SCHEMA_COLLECTIONS
GO
```

El resultado es:

Resultados		Mensajes				
	xml_collection_id	schema_id	principal_id	name	create_date	modify_date
1	1	4	NULL	sys	2009-04-13 12:59:13.390	2014-02-20 20:48:35.617
2	65536	6	NULL	AdditionalContactInfoSchemaCollection	2014-07-17 16:11:14.737	2014-07-17 16:11:14.750
3	65537	6	NULL	IndividualSurveySchemaCollection	2014-07-17 16:11:14.767	2014-07-17 16:11:14.767
4	65538	5	NULL	HRResumeSchemaCollection	2014-07-17 16:11:14.783	2014-07-17 16:11:14.783
5	65539	7	NULL	ProductDescriptionSchemaCollection	2014-07-17 16:11:14.823	2014-07-17 16:11:14.830
6	65540	7	NULL	ManuInstructionsSchemaCollection	2014-07-17 16:11:14.850	2014-07-17 16:11:14.850
7	65541	9	NULL	StoreSurveySchemaCollection	2014-07-17 16:11:14.863	2014-07-17 16:11:14.863

La siguiente consulta devuelve los nombres de colección de esquemas XML con el nombre de esquema relacional:

```
USE ADVENTUREWORKS2014
```

```
GO
```

```
SELECT XSC.XML_COLLECTION_ID,
       S.NAME + '.' + XSC.NAME AS XML_COLLECTION,
       XSC.PRINCIPAL_ID, XSC.CREATE_DATE, XSC.MODIFY_DATE
FROM SYS.XML_SCHEMA_COLLECTIONS XSC
JOIN SYS.SCHEMAS S
ON XSC.SCHEMA_ID = S.SCHEMA_ID
GO
```

El resultado sería:

Resultados		Mensajes			
	xml_collection_id	xml_collection	principal_id	create_date	modify_date
1	1	sys.sys	NULL	2009-04-13 12:59:13.390	2014-02-20 20:48:35.617
2	65536	Person.AdditionalContactInfoSchemaCollection	NULL	2014-07-17 16:11:14.737	2014-07-17 16:11:14.750
3	65537	Person.IndividualSurveySchemaCollection	NULL	2014-07-17 16:11:14.767	2014-07-17 16:11:14.767
4	65538	HumanResources.HRResumeSchemaCollection	NULL	2014-07-17 16:11:14.783	2014-07-17 16:11:14.783
5	65539	Production.ProductDescriptionSchemaCollection	NULL	2014-07-17 16:11:14.823	2014-07-17 16:11:14.830
6	65540	Production.ManuInstructionsSchemaCollection	NULL	2014-07-17 16:11:14.850	2014-07-17 16:11:14.850
7	65541	Sales.StoreSurveySchemaCollection	NULL	2014-07-17 16:11:14.863	2014-07-17 16:11:14.863

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.

En el ejemplo, se crea una tabla con campo XML asociado a un esquema XML (Person.IndividualSurveySchemaCollection).

El XML schema collection Person.IndividualSurveySchemaCollection, tiene este contenido:

Dirección URL: <http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey/IndividualSurvey.xsd>

```
<?xml version="1.0" encoding="UTF-8" ?>
- <xsd:schema targetNamespace="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey"
  xmlns="http://schemas.microsoft.com/sqlserver/2004/07/adventure-works/IndividualSurvey" elementFormDefault="qualified"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
- <xsd:annotation>
  <xsd:documentation>(c) 2008 Microsoft Corporation. All rights reserved. The following schema for Microsoft SQL Server
    informational purposes only. Microsoft Corporation ("Microsoft") may have trademarks, copyrights, or other intellectual
    property in the schema. Microsoft does not make any representation or warranty regarding the schema or any product or item
    provided to you on an AS IS basis. Microsoft disclaims all express, implied and statutory warranties, including but not
    limited to merchantability, fitness for a particular purpose, and freedom from infringement. Without limiting the generality of
    this warranty of any kind that any item developed based on the schema, or any portion of the schema, will not infringe a
    third party's intellectual property right of any person or entity in any country. It is your responsibility to seek licenses for such
    in the MICROSOFT SHALL NOT BE LIABLE FOR ANY DAMAGES OF ANY KIND ARISING OUT OF OR IN CONNECTION WITH THE USE OF THIS
    LIMITATION, ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL (INCLUDING ANY LOST PROFITS), PUNITIVE OR SPECIAL DAMAGES.
    HAS BEEN ADVISED OF SUCH DAMAGES.</xsd:documentation>
  </xsd:annotation>
- <xsd:simpleType name="SalaryType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="0-25000" />
    <xsd:enumeration value="25001-50000" />
    <xsd:enumeration value="50001-75000" />
    <xsd:enumeration value="75001-100000" />
    <xsd:enumeration value="greater than 100000" />
  </xsd:restriction>
  </xsd:simpleType>
- <xsd:simpleType name="MileRangeType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="0-1 Miles" />
    <xsd:enumeration value="1-2 Miles" />
    <xsd:enumeration value="2-5 Miles" />
    <xsd:enumeration value="5-10 Miles" />
    <xsd:enumeration value="10+ Miles" />
  </xsd:restriction>
  </xsd:simpleType>
- <xsd:element name="IndividualSurvey">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="TotalPurchaseYTD" type="xsd:decimal" minOccurs="0" maxOccurs="1" />
      <xsd:element name="DateFirstPurchase" type="xsd:date" minOccurs="0" maxOccurs="1" />
      <xsd:element name="BirthDate" type="xsd:date" minOccurs="0" maxOccurs="1" />
      <xsd:element name="MaritalStatus" type="xsd:string" minOccurs="0" maxOccurs="1" />
      <xsd:element name="YearlyIncome" type="SalaryType" minOccurs="0" maxOccurs="1" />
      <xsd:element name="Gender" type="xsd:string" minOccurs="0" maxOccurs="1" />
      <xsd:element name="TotalChildren" type="xsd:int" minOccurs="0" maxOccurs="1" />
      <xsd:element name="NumberChildrenAtHome" type="xsd:int" minOccurs="0" maxOccurs="1" />
      <xsd:element name="Education" type="xsd:string" minOccurs="0" maxOccurs="1" />
      <xsd:element name="Occupation" type="xsd:string" minOccurs="0" maxOccurs="1" />
      <xsd:element name="HomeOwnerFlag" type="xsd:string" minOccurs="0" maxOccurs="1" />
      <xsd:element name="NumberCarsOwned" type="xsd:int" minOccurs="0" maxOccurs="1" />
      <xsd:element name="Hobby" type="xsd:string" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="CommuteDistance" type="MileRangeType" minOccurs="0" maxOccurs="1" />
      <xsd:element name="Comments" type="xsd:string" minOccurs="0" maxOccurs="1" />
    </xsd:sequence>
  </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Ahora creamos la tabla, con un campo encuesta (survey) basado en este xml schema collection.

```
USE ADVENTUREWORKS2014
GO
```

```
CREATE TABLE TABLAPERSONSURVEY
```

```
(
  IDSURVEY INT PRIMARY KEY,
  SURVEYXML XML (PERSON.INDIVIDUALSURVEYSCHMACOLLECTION)
)
```



```
GO
```

XML sin tipo

El tipo de datos XML en SQL Server implementa la norma ISO estándar SQL2003 tipo de datos XML. Como tal, puede almacenar no sólo documentos en formato XML 1.0, sino también fragmentos de XML.

Algunos datos que no están con formato son rechazados. XML sin tipo es útil cuando no se sabe a priori el esquema. 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.

En el siguiente ejemplo se crea una tabla con columna o campo de tipo XML sin tipo.

```
USE ADVENTUREWORKS2014
```

```
GO
```

```
CREATE TABLE T1
```

```
(COL1 INT PRIMARY KEY,
```

```
COL2 XML)
```

```
GO
```

Variables XML

Ejemplo de una variable de tipo XML

```
DECLARE @X XML
```

Ejemplo de una variable de tipo XML asignando un schema collection.

```
DECLARE @X XML (SALES.STORESURVEYSCHMACOLLECTION)
```

```
GO
```

7.1.3 FOR XML y mejoras OPEN XML.

FOR XML

Una consulta SELECT devuelve resultados como un conjunto de filas. Puede recuperar opcionalmente resultados formales de una consulta SQL como XML mediante la especificación de la cláusula de XML para la consulta. La cláusula FOR XML se puede utilizar en las consultas de alto nivel y en las subconsultas. En alto nivel cláusula FOR XML se puede utilizar sólo en la instrucción SELECT. En las subconsultas, FOR XML se puede utilizar en el INSERT, UPDATE y DELETE. También se puede utilizar en instrucciones de asignación.

La cláusula FOR XML, se especifica uno de estos modos:

- RAW
- AUTO
- EXPLICIT
- PATH

El modo RAW genera un único elemento <row> por cada fila del conjunto de filas que devuelve la instrucción SELECT. Puede generar jerarquía XML escribiendo anidada consultas FOR XML.

El modo AUTO genera anidando en el XML resultante mediante el uso de la heurística basada en la forma en que se especifica la instrucción SELECT. Usted tiene control mínimo sobre la forma del XML generado. La anidada consulta FOR XML se puede escribir para generar jerarquía XML XML allá de la forma que se genera por la heurística del modo AUTO.

El modo EXPLICIT permite más control sobre la forma de la XML. Usted puede mezclar los atributos y elementos a voluntad para decidir la forma del XML. Se requiere un formato específico para el conjunto de filas resultante que se genera debido a la ejecución de la consulta. Este formato de conjunto de filas se asigna a continuación, en forma XML. El poder de modo explícito es mezclar atributos y elementos a voluntad, crear contenedores y propiedades complejas anidadas, crear valores separados por espacios (por ejemplo, IDPedido atributo puede tener una lista de los valores de ID de pedido), y el contenido mixtos.

Sin embargo, escribir consultas en modo explícito puede ser engorroso. Puede utilizar algunas de las nuevas opciones de XML, tales como FOR XML RAW / AUTO / PATH y la directiva TYPE, en lugar de utilizar el modo explícito para generar las jerarquías.

El modo de PATH provee para la capacidad de consulta XML, proporciona la flexibilidad del modo EXPLICIT de una manera más simple.

Estos modos son en efecto sólo para la ejecución de la consulta para la que se establecen. No afectan a los resultados de las consultas posteriores.

OPEN XML

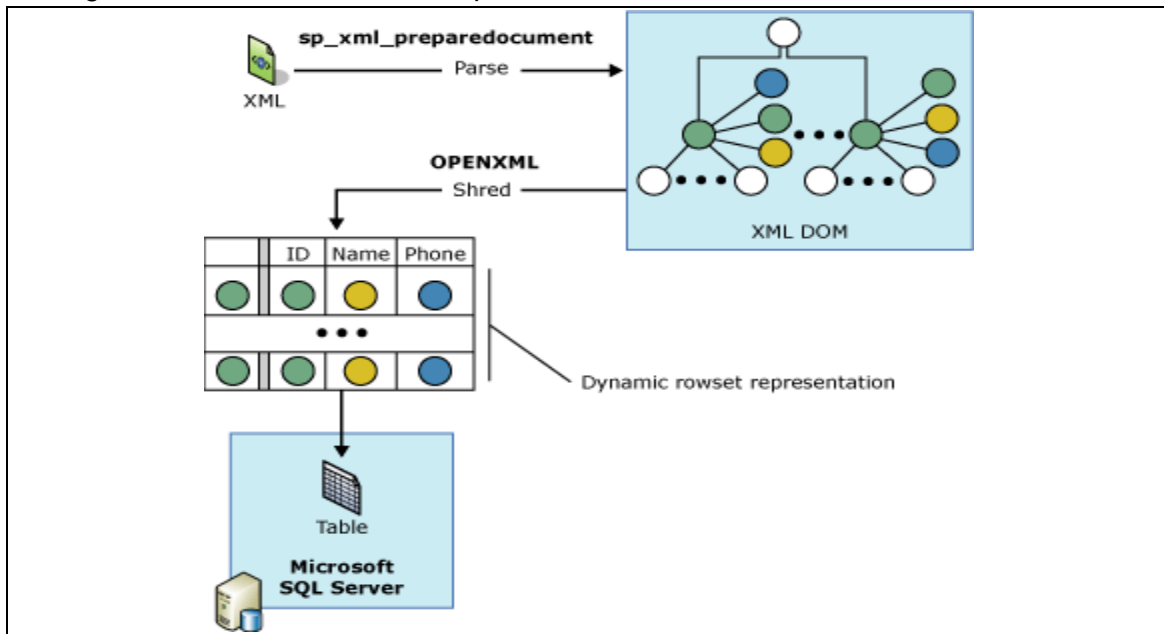
OPENXML, proporciona un conjunto de filas en memoria sobre documentos XML que es similar a una tabla o una vista. OPENXML permite el acceso a datos XML como si se trata de un conjunto de filas relacional. Para ello, proporciona una visión de conjunto de filas de la representación interna de un documento XML. Los registros del conjunto de filas se pueden almacenar en tablas de bases de datos.

OPENXML se puede utilizar en SELECT y SELECT INTO, donde los orígenes del conjunto de filas sean una vista u OPENROWSET.

Para escribir consultas en un documento XML mediante el uso de OPENXML, primero debe llamar sp_xml_preparedocument. Este analiza el documento XML y devuelve un identificador para el documento analizado que está listo para el consumo. El documento analizado es una representación árbol del modelo de objetos de documento (DOM) de varios nodos en el documento XML. El documento se pasa a OPENXML. OPENXML a continuación, proporciona una vista de conjunto de filas del documento, en base de los parámetros pasados a la misma.

La representación interna de un documento XML debe ser removido de la memoria mediante una llamada al procedimiento almacenado `sp_xml_removedocument` del sistema para liberar la memoria.

La siguiente ilustración muestra el proceso.



7.2 PROCESAMIENTO XML EN SQL SERVER

7.2.1 Almacenamiento de datos XML.

El siguiente ejemplo, almacena datos en una tabla con campo XML sin tipo.

```
USE ADVENTUREWORKS2014
```

```
GO
```

```
DECLARE @XML AS XML
```

```
SET @XML = '<CLIENTES> <CLIENTE>
            <CLIENTEID>C01</CLIENTEID>
            <NOMBRE>JUAN</NOMBRE>
          </CLIENTE>
          <CLIENTE>
            <CLIENTEID>C02</CLIENTEID>
            <NOMBRE>JORGE</NOMBRE>
          </CLIENTE>
          <CLIENTE>
            <CLIENTEID>C03</CLIENTEID>
            <NOMBRE>CARLOS</NOMBRE>
          </CLIENTE>
        </CLIENTES>'
```

```
INSERT INTO T1
```

```
VALUES
```

```
('1', @XML)
```

```
GO
```

Si consultamos la tabla, tenemos el siguiente resultado:

```
SELECT * FROM T1
GO
```

	Col1	Col2
1	1	<Clientes><Cliente><ClienteID>C01</ClienteID><No...

Y si hacemos click en el dato de la columna 2, visualizamos los siguiente:

Col25.xml	http://schemas.mic...dividualSurvey.xsd
<pre> <Clientes> <Cliente> <ClienteID>C01</ClienteID> <Nombre>Juan</Nombre> </Cliente> <Cliente> <ClienteID>C02</ClienteID> <Nombre>Jorge</Nombre> </Cliente> <Cliente> <ClienteID>C03</ClienteID> <Nombre>Carlos</Nombre> </Cliente> </Clientes> </pre>	

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:\XMLFiles\XMLClientes.xml, como un BLOB mediante el uso de OPENROWSET.

Una nueva fila se insertará en la tabla denominada T1 con un valor de 2 para la clave principal y el contenido del archivo en la columna XML llamada Detalle.

```
INSERT INTO T1
SELECT 2, DETALLE
FROM (SELECT *
      FROM OPENROWSET
        (BULK 'C:\XMLFILES\XMLClientes.XML',
         SINGLE_BLOB)
      AS DETALLE) AS R(DETALLE)
GO
```

Ahora si consultamos la tabla tenemos el siguiente resultado:

```
SELECT * FROM T1
GO
```

	Col1	Col2
1	1	<Clientes><Cliente><ClienteID>C01</ClienteID><No...
2	2	<Clientes><Cliente><ClienteID>C04</ClienteID><No...

7.2.2 Recuperación de 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.

7.2.2.1 Usar modo RAW.

Es el modo más básico.

No incluye Root.

Formatos en los atributos.

Sin anidación.

Muestre el nombre, apellidos y título de Person en XML, sin considerar Root (raíz).

```
USE ADVENTUREWORKS2014
GO
```

```
SELECT    FIRSTNAME,
          LASTNAME,
          TITLE
FROM PERSON.PERSON
FOR XML RAW
GO
```

Se muestra:

Resultados		Mensajes
XML_F52E2B61-18A1-11d1-B105-00805F49916B		
1	<row FirstName="Ken" LastName="Sánchez"/><row Fi...	

Al hacer click en el link, nos muestra el siguiente resultado XML.

```
XML_F52E2B61-18A...00805F49916B3.xml X XML_F52E2B61-18A...00805F49916B2.xml
<row FirstName="Ken" LastName="Sánchez" />
<row FirstName="Terri" LastName="Duffy" />
<row FirstName="Roberto" LastName="Tamburello" />
<row FirstName="Rob" LastName="Walters" />
<row FirstName="Gail" LastName="Erickson" Title="Ms." />
<row FirstName="Jossef" LastName="Goldberg" Title="Mr." />
<row FirstName="Dylan" LastName="Miller" />
<row FirstName="Diane" LastName="Margheim" />
<row FirstName="Gigi" LastName="Matthew" />
<row FirstName="Michael" LastName="Raheem" />
<row FirstName="Ovidiu" LastName="Cracium" />
<row FirstName="Thierry" LastName="D'Hers" />
<row FirstName="Janice" LastName="Galvin" Title="Ms." />
<row FirstName="Michael" LastName="Sullivan" />
```

La misma consulta anterior, con la opción elements.

```
USE ADVENTUREWORKS2014
GO
```

```
SELECT    FIRSTNAME,
```

```

        LASTNAME,
        TITLE
FROM PERSON.PERSON
FOR XML RAW, ELEMENTS
GO

```

Al hacer click en el enlace del resultado, se muestra lo siguiente:

```

XML_F52E2B61-18A...0805F49916B11.xml X SQLQuery1.sql - SA...Works2014 (sa (52))*
<row>
  <FIRSTNAME>Ken</FIRSTNAME>
  <LASTNAME>Sánchez</LASTNAME>
</row>
<row>
  <FIRSTNAME>Terri</FIRSTNAME>
  <LASTNAME>Duffy</LASTNAME>
</row>

```

Modificando la consulta anterior, renombrando la etiqueta ROW.

```

USE ADVENTUREWORKS2014
GO

SELECT    FIRSTNAME,
          LASTNAME,
          TITLE
FROM PERSON.PERSON
FOR XML RAW ('REGISTROPERSONA'), ELEMENTS
GO

```

Al hacer click en el enlace del resultado, se muestra lo siguiente:

```

XML_F52E2B61-18A...00805F49916B5.xml X SQLQuery1.sql - SA...Works2014
<RegistroPersona>
  <FirstName>Ken</FirstName>
  <LastName>Sánchez</LastName>
</RegistroPersona>
<RegistroPersona>
  <FirstName>Terri</FirstName>
  <LastName>Duffy</LastName>
</RegistroPersona>

```

Modificando la consulta anterior, con la opción Root

```

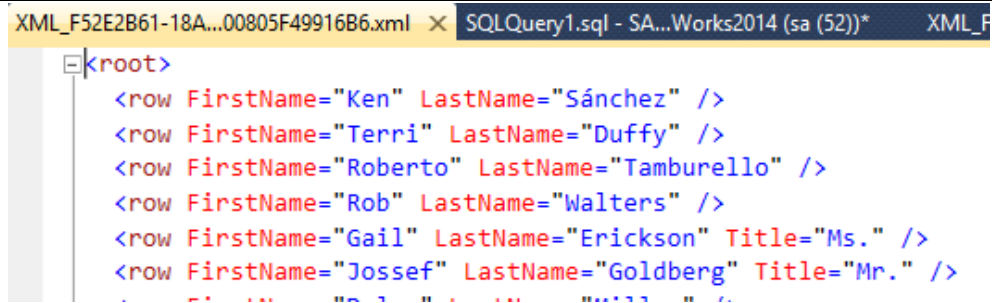
USE ADVENTUREWORKS2014
GO

SELECT    FIRSTNAME,
          LASTNAME,
          TITLE
FROM PERSON.PERSON

```

```
FOR XML RAW, ROOT
GO
```

Al hacer click en el enlace, el resultado será:



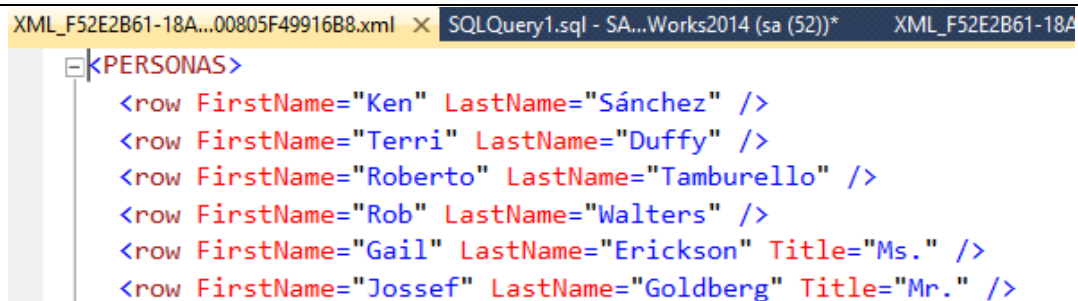
```
XML_F52E2B61-18A...00805F49916B6.xml X SQLQuery1.sql - SA...Works2014 (sa (52))* XML_F
<root>
  <row FirstName="Ken" LastName="Sánchez" />
  <row FirstName="Terri" LastName="Duffy" />
  <row FirstName="Roberto" LastName="Tamburello" />
  <row FirstName="Rob" LastName="Walters" />
  <row FirstName="Gail" LastName="Erickson" Title="Ms." />
  <row FirstName="Jossef" LastName="Goldberg" Title="Mr." />
```

Modificando la consulta anterior, con la opción Root, etiquetando con Personas.

```
USE ADVENTUREWORKS2014
GO

SELECT    FIRSTNAME,
          LASTNAME,
          TITLE
FROM PERSON.PERSON
FOR XML RAW, ROOT ('PERSONAS')
GO
```

Al hacer click en el enlace, el resultado será:



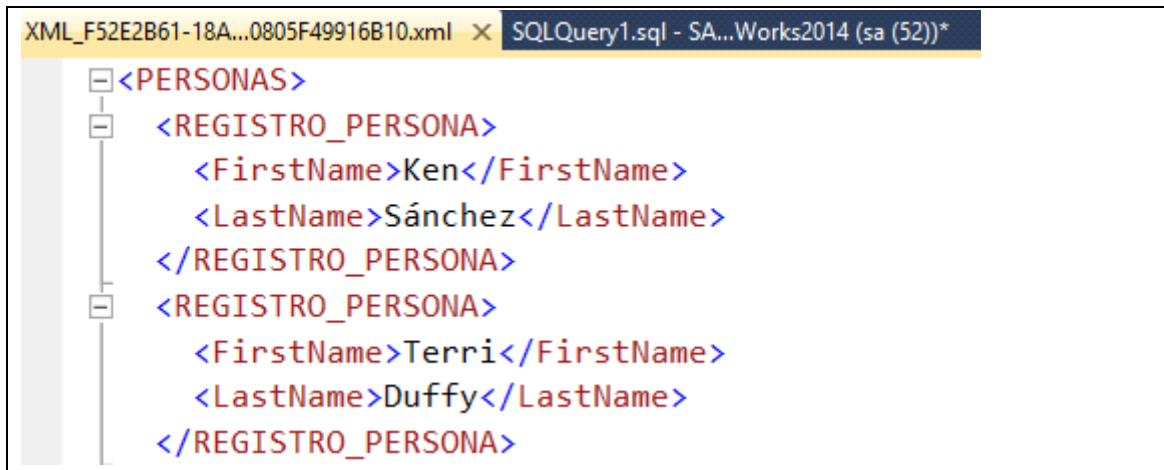
```
XML_F52E2B61-18A...00805F49916B8.xml X SQLQuery1.sql - SA...Works2014 (sa (52))* XML_F52E2B61-18A
<PERSONAS>
  <row FirstName="Ken" LastName="Sánchez" />
  <row FirstName="Terri" LastName="Duffy" />
  <row FirstName="Roberto" LastName="Tamburello" />
  <row FirstName="Rob" LastName="Walters" />
  <row FirstName="Gail" LastName="Erickson" Title="Ms." />
  <row FirstName="Jossef" LastName="Goldberg" Title="Mr." />
```

Modificando la consulta anterior, para incluir todas las opciones, etiquetando a la fila (Row) y a Root.

```
USE ADVENTUREWORKS2014
GO

SELECT    FIRSTNAME,
          LASTNAME,
          TITLE
FROM PERSON.PERSON
FOR XML RAW ('REGISTRO_PERSONA'), ELEMENTS, ROOT ('PERSONAS')
GO
```

Al hacer click en el enlace, el resultado será:



7.2.2.2 Usar modo AUTO.

No incluye Root

Formatos en los atributos

Los nombres de las filas se basan en las columnas de las tablas de sentencia Select

Contiene la segunda tabla dentro del primer elemento de la primera tabla

Puede agrupar XML con Order By

Opción de etiquetar no disponible

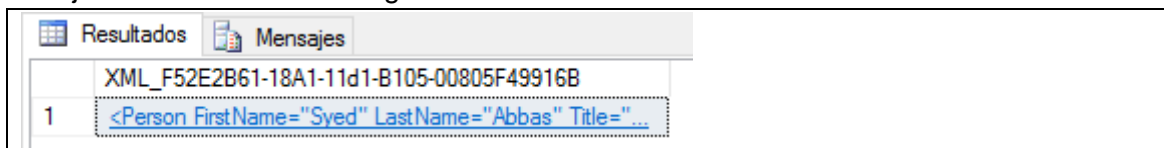
Muestre el nombre, apellido y título de la persona, así también la dirección de correo. Para ello, se combinará 02 tablas, donde la primera tabla es Person, que contendrá elementos de la segunda tabla, usando FOR XML modo AUTO.

USE ADVENTUREWORKS2014

GO

```
SELECT PERSON.FIRSTNAME, PERSON.LASTNAME,
        PERSON.TITLE, EMAIL.EMAILADDRESS
FROM PERSON.PERSON PERSON JOIN
        PERSON.EMAILADDRESS EMAIL
        ON PERSON.BUSINESSENTITYID = EMAIL.BUSINESSENTITYID
ORDER BY PERSON.LASTNAME
FOR XML AUTO
GO
```

Al ejecutar se mostrará el siguiente resultado:



Al hacer click en el enlace, se mostrará en formato XML.


```

XML_F52E2B61-18A...0805F49916B12.xml  X SQLQuery1.sql - SA...Works2014 (sa (52))*  XML_F52E2B61-18A...00805
<Person FirstName="Syed" LastName="Abbas" Title="Mr.">
  <Email EmailAddress="syed0@adventure-works.com" />
</Person>
<Person FirstName="Catherine" LastName="Abel" Title="Ms.">
  <Email EmailAddress="catherine0@adventure-works.com" />
</Person>
<Person FirstName="Kim" LastName="Abercrombie" Title="Ms.">
  <Email EmailAddress="kim2@adventure-works.com" />
</Person>
<Person FirstName="Kim" LastName="Abercrombie">
  <Email EmailAddress="kim1@adventure-works.com" />
  <Email EmailAddress="kim7@adventure-works.com" />
</Person>

```

Modificar la consulta anterior, agregando la opción ELEMENTS.

```

USE ADVENTUREWORKS2014
GO

SELECT     PERSON.FIRSTNAME, PERSON.LASTNAME,
           PERSON.TITLE, EMAIL.EMAILADDRESS
FROM PERSON.PERSON PERSON JOIN
      PERSON.EMAILADDRESS EMAIL
      ON PERSON.BUSINESSENTITYID = EMAIL.BUSINESSENTITYID
ORDER BY PERSON.LASTNAME
FOR XML AUTO, ELEMENTS
GO

```

Al hacer click en el enlace, se mostrará lo siguiente:

Resultados Mensajes

XML_F52E2B61-18A1-11d1-B105-00805F49916B
1 <Person><FirstName>Syed</FirstName><LastName>Abb...

```

XML_F52E2B61-18A...0805F49916B13.xml  X SQLQuery1.sql - SA...Works2014 (sa (52))*  XML_F52E2B61-18A...00805F4991
<Person>
  <FirstName>Syed</FirstName>
  <LastName>Abbas</LastName>
  <Title>Mr.</Title>
  <Email>
    <EmailAddress>syed0@adventure-works.com</EmailAddress>
  </Email>
</Person>
<Person>
  <FirstName>Catherine</FirstName>
  <LastName>Abel</LastName>
  <Title>Ms.</Title>
  <Email>
    <EmailAddress>catherine0@adventure-works.com</EmailAddress>
  </Email>
</Person>

```

Modificar la consulta anterior, agregando la opción ROOT.

```
USE ADVENTUREWORKS2014
GO

SELECT      PERSON.FIRSTNAME, PERSON.LASTNAME,
            PERSON.TITLE, EMAIL.EMAILADDRESS
FROM PERSON.PERSON PERSON JOIN
            PERSON.EMAILADDRESS EMAIL
            ON PERSON.BUSINESSENTITYID = EMAIL.BUSINESSENTITYID
ORDER BY PERSON.LASTNAME
FOR XML AUTO, ROOT
GO
```

Al hacer click en el enlace, se mostrará lo siguiente:

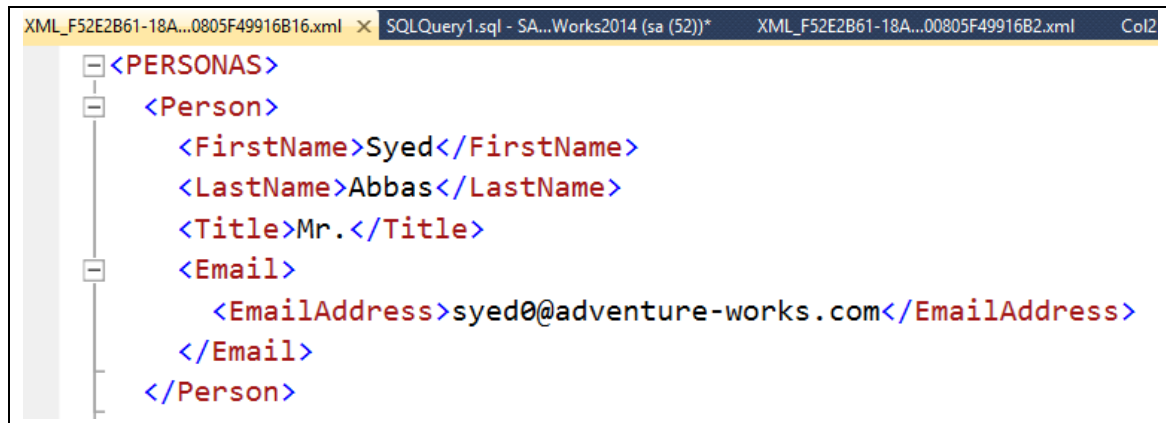
```
<root>
  <Person FirstName="Syed" LastName="Abbas" Title="Mr.">
    <Email EmailAddress="syed0@adventure-works.com" />
  </Person>
  <Person FirstName="Catherine" LastName="Abel" Title="Ms.">
    <Email EmailAddress="catherine0@adventure-works.com" />
  </Person>
  <Person FirstName="Kim" LastName="Abercrombie" Title="Ms.">
    <Email EmailAddress="kim2@adventure-works.com" />
  </Person>
  <Person FirstName="Kim" LastName="Abercrombie">
    <Email EmailAddress="kim1@adventure-works.com" />
    <Email EmailAddress="kim7@adventure-works.com" />
  </Person>
```

Modificar la consulta anterior, agregando la opción ELEMENTS y ROOT con etiqueta PERSONAS.

```
USE ADVENTUREWORKS2014
GO

SELECT      PERSON.FIRSTNAME, PERSON.LASTNAME,
            PERSON.TITLE, EMAIL.EMAILADDRESS
FROM PERSON.PERSON PERSON JOIN
            PERSON.EMAILADDRESS EMAIL
            ON PERSON.BUSINESSENTITYID = EMAIL.BUSINESSENTITYID
ORDER BY PERSON.LASTNAME
FOR XML AUTO, ELEMENTS, ROOT ('PERSONAS')
GO
```

Al hacer click en el enlace, se mostrará lo siguiente:



7.2.2.3 Usar modo EXPLICIT.

En un modo explícito, la escritura de la consulta controla la forma del documento XML devuelto por la ejecución de la misma. La consulta debe ser escrita de una manera específica para que la información adicional acerca de la anidación sea explícitamente como parte de la consulta. También puede especificar configuraciones adicionales a nivel de columna usando las directivas. Cuando se especifica el modo explícito, usted debe asumir la responsabilidad de asegurar que el XML generado está bien formado y válido (en el caso de un esquema XML-DATA).

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.

En el siguiente ejemplo, vamos a listar los datos de los clientes (CustomerID) y las órdenes generadas por cada uno.

Primero se elabora la primera consulta (de la tabla Customers) con tag 1 y parent 0, CustomerID y null para la Orden. Luego se une la consulta combinada (Customers y Orders) con tag 2, parent con valor 1, CustomerId y OrderId. Todo esto ordenado por CustomerId y OrderId.

```
USE NORTHWIND
GO
```

```
SELECT      1                                AS TAG,
```

```

        NULL
        CUSTOMERS.CUSTOMERID
        NULL
    AS PARENT,
    AS [CUSTOMER!1!CUSTOMERID],
    AS [ORDER!2!ORDERID]
FROM CUSTOMERS
    UNION ALL
SELECT
    2,
    1,
    CUSTOMERS.CUSTOMERID,
    ORDERS.ORDERID
FROM CUSTOMERS, ORDERS
WHERE CUSTOMERS.CUSTOMERID = ORDERS.CUSTOMERID
ORDER BY [CUSTOMER!1!CUSTOMERID], [ORDER!2!ORDERID]
GO

```

A continuación se muestra el resultado:

Resultados		Mensajes		
	Tag	Parent	Customer!1!CustomerID	Order!2!OrderID
1	1	NULL	ALFKI	NULL
2	2	1	ALFKI	10643
3	2	1	ALFKI	10692
4	2	1	ALFKI	10702
5	2	1	ALFKI	10835
6	2	1	ALFKI	10952
7	2	1	ALFKI	11011
8	1	NULL	ANATR	NULL
9	2	1	ANATR	10308
10	2	1	ANATR	10625
11	2	1	ANATR	10759

Ahora modificamos esta consulta colocando la opción FOR XML EXPLICIT para generar un resultado en formato XML y de manera mas resumida, esto porque se definió de forma explícita en la consulta el formato del documento XML.

```

USE NORTHWIND
GO

SELECT
    1
    NULL
    CUSTOMERS.CUSTOMERID
    NULL
    AS TAG,
    AS PARENT,
    AS [CUSTOMER!1!CUSTOMERID],
    AS [ORDER!2!ORDERID]
FROM CUSTOMERS
    UNION ALL
SELECT
    2,
    1,
    CUSTOMERS.CUSTOMERID,
    ORDERS.ORDERID
FROM CUSTOMERS, ORDERS
WHERE CUSTOMERS.CUSTOMERID = ORDERS.CUSTOMERID

```

```
ORDER BY [CUSTOMER!1!CUSTOMERID], [ORDER!2!ORDERID]
FOR XML EXPLICIT
GO
```

El resultado es:

Resultados	Mensajes
XML_F52E2B61-18A1-11d1-B105-00805F49916B	
1	<Customer CustomerID="ALFKI"><Order OrderID="106...

Al hacer click en el enlace se mostrará en formato XML.

XML_F52E2B61-18A...0805F49916B20.xml	XML_F52E2B61-18A...0805F49916B...
<pre><Customer CustomerID="ALFKI"> <Order OrderID="10643" /> <Order OrderID="10692" /> <Order OrderID="10702" /> <Order OrderID="10835" /> <Order OrderID="10952" /> <Order OrderID="11011" /> </Customer> <Customer CustomerID="ANATR"> <Order OrderID="10308" /> <Order OrderID="10625" /> <Order OrderID="10759" /> <Order OrderID="10926" /> </Customer></pre>	

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 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 trabajar 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.

Bibliografía

ROSS, Mistry & STACIA, Misner

2014 Introducing Microsoft SQL Server 2014 (Technical Overview)
Redmond, Washington : Microsoft Press
(978.0.7356.8475.1)

RODNEY, Landrum

2009 SQL Server Tacklebox
Redgate
Free Ebook (978.1.9064.34.25.0)

MICROSOFT DEVELOPER NETWORK

2014 Microsoft SQL Server 2014
[https://msdn.microsoft.com/es-es/library/ms130214\(v=sql.120\).aspx](https://msdn.microsoft.com/es-es/library/ms130214(v=sql.120).aspx)

MICROSOFT VIRTUAL ACADEMY

2016 SQL Server 2016
<http://www.microsoftvirtualacademy.com/ebooks>

DBLEARNER

2014 ¿Cómo garantizar la integridad de los Backups?
<http://dblearner.com/tag/administracion/>

MICROSOFT TECHNET

2014 SQLXML 4.0 Programming Concepts
[https://technet.microsoft.com/en-us/library/ms171779\(v=sql.120\).aspx](https://technet.microsoft.com/en-us/library/ms171779(v=sql.120).aspx)

WINDY MARTIN

2014 Training Video FOR XML
<https://www.youtube.com/user/DawaBITrainingVideos/videos>