

Capítulo 5

Introducción al lenguaje SQL

Al finalizar el capítulo, el alumno podrá:

- Identificar la sintaxis del lenguaje SQL de base de datos.
- Aplicar las sentencias de definición de datos (DDL).
- Establecer las diferencias entre SQL Server 2016 y SQL.
- Diferenciar los dialectos del lenguaje según el manejador de base de datos.

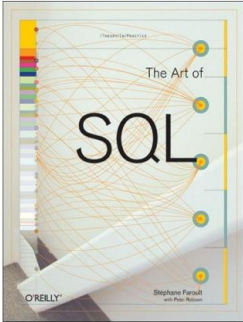
Temas:

1. Definición de SQL
2. SQL-Server y Transact SQL
3. Implementando la estructura de una base de datos con Transact SQL
4. Importando datos desde diversas fuentes de información

1. Definición de SQL

Definición del lenguaje SQL
¿Qué es Structured Query Language o SQL?

- Es un lenguaje de gestión de datos de base de datos relacionales.
- No es un software, sino un dialecto con reglas gramaticales.
- Es declarativo y permite especificar diversas operaciones.



5 - 4

Copyright © Todos los Derechos Reservados - Cibertec Perú S.A.C.

1.1. Introducción

El lenguaje estructurado de consulta (**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, lo cual admite 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).

El SQL es un lenguaje de base de datos normalizado, utilizado por los diferentes motores de bases de datos relacionales (SQL Server, MS Access, ORACLE, Informix, DB2, My SQL, entre otros) para realizar determinadas operaciones sobre los datos o sobre la estructura de los mismos, pero como sucede con cualquier sistema de normalización, hay excepciones para casi todo; de hecho, cada motor de bases de datos tiene sus peculiaridades y lo hace diferente de otro motor, por lo tanto, el lenguaje SQL normalizado (ANSI) no servirá para resolver todos los problemas, aunque sí puede asegurar que cualquier sentencia escrita en SQL ANSI será interpretable por cualquier motor de datos.

1.2. Clasificación de lenguaje SQL

Las instrucciones SQL se clasifican según su propósito en tres grupos:

- El DDL (Data Description Language), es la parte del SQL dedicada a la definición de la base de datos, consta de sentencias para **definir la estructura de la base de datos**, permiten crear la base de datos, crear, modificar o eliminar la estructura de las tablas, crear índices, definir reglas de validación de datos, relaciones entre las tablas, etc. Permite definir gran parte del nivel

interno de la base de datos. Por este motivo estas sentencias serán utilizadas normalmente por el administrador de base de datos.

Comando	Descripción
CREATE	Utilizado para crear nuevas tablas, campos e índices.
DROP	Empleado para eliminar tablas e índices.
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

- El DML (Data Manipulation Language), se compone de las instrucciones para el manejo de los datos, para insertar nuevos datos, modificar datos existentes, para eliminar datos y la más utilizada, para recuperar datos de la base de datos. Veremos que una sola instrucción de recuperación de datos es tan potente que permite recuperar datos de varias tablas a la vez, realizar cálculos sobre estos datos y obtener resúmenes.

El DML interactúa con el nivel externo de la base de datos por lo que sus instrucciones son muy parecidas, por no decir casi idénticas, de un sistema a otro, el usuario solo indica lo que quiere recuperar no cómo se tiene que recuperar, no influye el cómo están almacenados los datos.

Es el lenguaje que utilizan los programadores y los usuarios de la base de datos.

Comando	Descripción
SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado.
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados.
DELETE	Utilizado para eliminar registros de una tabla de una base de datos.

- El DCL (Data Control Language) se compone de instrucciones que permiten ejercer un control sobre los datos tal como la asignación de privilegios de acceso a los datos.

Comando	Descripción
GRANT	Utilizado para conceder privilegios de acceso a usuarios.
REVOKE	Utilizado para suprimir privilegios de acceso a usuarios

- El TCL (Transaction Control Language) se compone de instrucciones que permiten controlar y gestionar transacciones para mantener la integridad de datos dentro de las sentencias SQL.

Comando	Descripción
BEGIN TRAN	Inicio de una transacción
COMMIT TRAN	Confirmar la transacción completa.
ROLLBACK TRAN	Vuelve al principio si algo no esta bien en la transacción.

**Para recordar:**

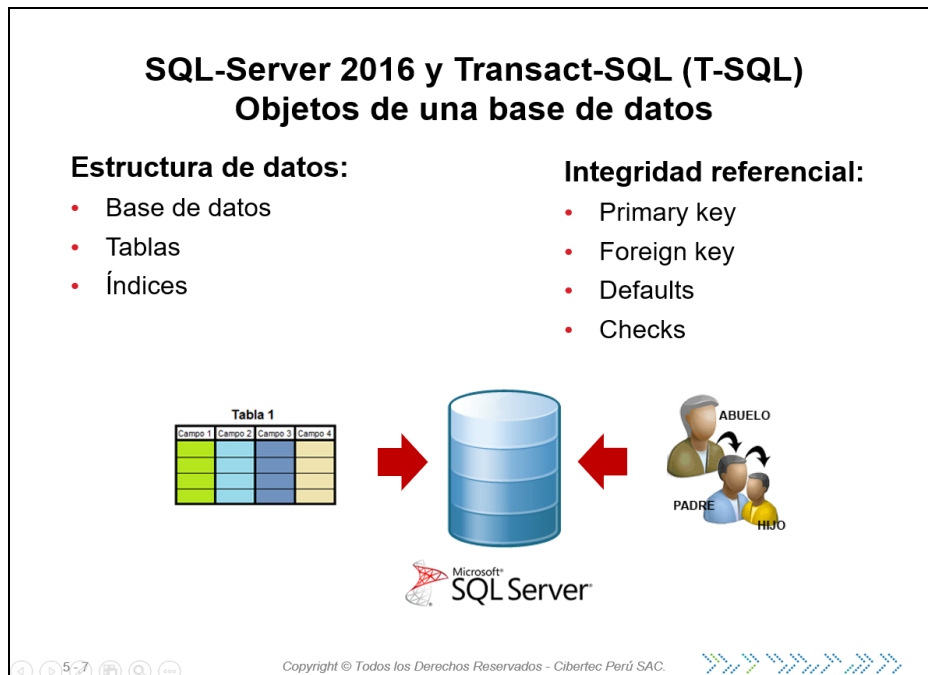
La historia de SQL empieza en 1974 con la definición, por parte de Donald Chamberlin y de otras personas que trabajaban en los laboratorios de investigación de IBM, de un lenguaje para la especificación de las características de las bases de datos que adoptaban el modelo relacional. Este lenguaje se llamaba **SEQUEL** (Structured English Query Language) y se implementó en un prototipo llamado SEQUEL-XRM entre 1974 y 1975. Las experimentaciones con ese prototipo condujeron, entre 1976 y 1977, a una revisión del lenguaje (SEQUEL/2), que a partir de ese momento cambió de nombre por motivos legales, convirtiéndose en SQL. El prototipo (System R), basado en este lenguaje, se adoptó y utilizó internamente en IBM. Además, lo adoptaron algunos de sus clientes elegidos.

Gracias al éxito de este sistema, que no estaba todavía comercializado, también otras compañías empezaron a desarrollar sus productos relacionales basados en SQL. A partir de 1981, IBM comenzó a entregar sus productos relacionales y en 1983 empezó a vender DB2. En el curso de los años ochenta, numerosas compañías (por ejemplo: Oracle y Sybase, solo por citar algunos) comercializaron productos basados en SQL, que se convierte en el estándar industrial de hecho, por lo que respecta a las bases de datos relacionales.

Extraído de:

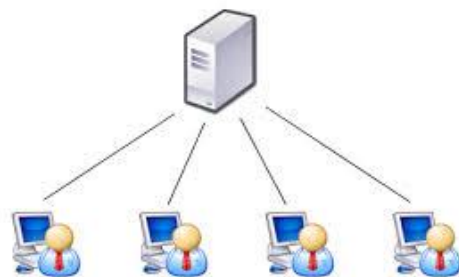
<http://personal.lobocom.es/claudio/sql001.htm>

2. SQL-Server 2016 y Transact-SQL (T-SQL)



2.1 SQL Server 2016

SQL Server es un sistema administrador para bases de datos relacionales basadas en la arquitectura cliente / servidor (RDBMS) que usa Transact-SQL para mandar peticiones entre un cliente y el SQL Server.



SQL Server usa la arquitectura cliente / servidor para separar la carga de trabajo, en tareas que corran en computadoras tipo servidor y tareas que corran en computadoras tipo cliente:

- El cliente es responsable de la parte lógica y de presentar la información al usuario. Generalmente, el cliente corre en una o más computadoras cliente, aunque también puede correr en una computadora servidor con SQL Server.
- SQL Server administra las bases de datos y distribuye los recursos disponibles del servidor (tales como memoria, operaciones de disco, etc.) entre las múltiples peticiones.

La arquitectura cliente /servidor permite desarrollar aplicaciones para realizar en una variedad de ambientes.



Para recordar:

El SQL Server como RDBMS es responsable de:

- *Mantener las relaciones entre la información y la base de datos.*
- *Asegurarse de que la información es almacenada correctamente, es decir, que las reglas que definen las relaciones entre los datos no sean violadas.*
- *Recuperar toda la información en un punto conocido, en caso de que el sistema falle.*

2.2 Transact - SQL

Es una versión de SQL (**Structured Query Language**) usado como lenguaje de programación para SQL Server. Es un conjunto de comandos que admite especificar la información que se desea restaurar o modificar. Además, permite tener acceso a la información, realizar búsquedas, actualizar y administrar sistemas de bases de datos relacionales.

SQL brinda muchas más opciones, ya que permite utilizar más funciones que el DBMS (**Database Management System** - Sistema de gestión de base de datos). Por ejemplo:

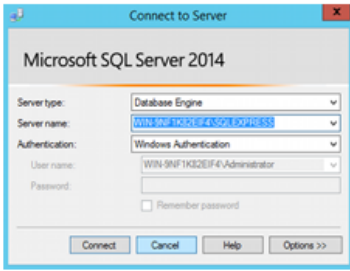
- **Definición de datos**
Permite definir la estructura y organización de datos almacenados y de las relaciones entre ellos.
- **Recuperación de datos**
Permite recuperar datos almacenados de la base de datos y utilizarlos.
- **Manipulación de datos**
Permite actualizar la base de datos, añadiendo nuevos datos, eliminando datos y modificando datos almacenados.
- **Control de acceso**
Es utilizado para restringir la capacidad de un usuario para recuperar, añadir y modificar datos, protegiendo los datos almacenados frente a accesos no autorizados.
- **Compartición de datos**
Se utiliza para coordinar la compartición de datos por parte de usuarios concurrentes, asegurando que no interfieran unos con otros.
- **Integridad de datos**
SQL define restricciones de integridad en la base datos, protegiéndola contra corrupciones indebidas o actualizaciones inconsistentes.


3. Implementando la estructura de una base de datos con lenguaje T- SQL

Implementando la estructura de base de datos con lenguaje T-SQL

```
CREATE DATABASE [nombre]
ON
PRIMARY (NAME = DATA,
FILENAME = 'C:\DATA.MDF',
SIZE = 5,
MAXSIZE = 50,
FILEGROWTH = 15%)
LOG ON (NAME = LOG,
FILENAME = 'C:\LOG.LDF',
SIZE = 2,
MAXSIZE = 10,
FILEGROWTH = 15%)
GO
```

```
DROP DATABASE [nombre]
```



5 - 8Copyright © Todos los Derechos Reservados - Cibertec Perú SAC.

3.1 Creación de una base de datos

Para crear una base de datos, se determina el nombre de la base de datos, el propietario (el usuario que crea la base de datos), su tamaño y los archivos, así como, grupos de archivos utilizados para almacenarla.

No obstante, antes de crear una base de datos, se debe considerar lo siguiente:

- El usuario que crea la base de datos se convierte en su propietario.
- En un servidor pueden crearse hasta 32.767 bases de datos.
- El nombre de la base de datos debe ajustarse a las reglas establecidas para los identificadores.

De forma predeterminada, tienen permiso para crear una base de datos, las funciones fijas del servidor **sysadmin** y **dbowner**, aunque se puede otorgar permisos a otros usuarios.

Se utilizan tres tipos de archivos para almacenar una base de datos:

A. Archivo principal

Contiene la información de inicio para la base de datos. Es utilizado también para almacenar datos. Cada base de datos tiene un único archivo principal. La extensión recomendada para los nombres de archivos de datos principales es **.mdf**.

B. Archivo secundario

Los archivos de datos secundarios son opcionales, están definidos por el usuario y almacenan los datos del usuario. Se pueden utilizar para distribuir datos en varios discos colocando cada archivo en una unidad de disco distinta. La extensión de nombre de archivo recomendada para archivos de datos secundarios es **.ndf**.

C. Registro de transacciones

Estos archivos contienen la información de registro que se utiliza para recuperar la base de datos. Debe haber al menos, un archivo de registro de transacciones para cada base de datos, aunque puede haber más de uno. El tamaño mínimo para un archivo de registro es 512 kilobytes (KB). La extensión recomendada para los nombres de archivos de registro es **.ldf**.

Cuando se crea una base de datos, todos los archivos que la componen se llenan con ceros que suplantando los datos ya eliminados que hubieran quedado en el disco. Aunque esto provoque que el proceso de creación de los archivos sea más largo, para evitar que el sistema operativo tenga que llenar los archivos con ceros cuando se escriban por primera vez datos durante las operaciones habituales con la base de datos.

Además, es recomendable especificar el tamaño máximo de crecimiento del archivo, para evitar que se agote el espacio disponible en el disco al agregar datos. Asimismo, para especificar un tamaño máximo para el archivo, se debe utilizar el parámetro **MAXSIZE** de la instrucción **CREATE DATABASE** o bien la opción **Limitar crecimiento de archivo a (MB)**, cuando se utilice el cuadro de diálogo **Propiedades del administrador corporativo** de SQL Server para crear la base de datos.

Después de crear una base de datos, se recomienda crear una copia de seguridad de la base de datos máster.

3.1.1 Grupos de archivos en una base de datos

Los grupos de archivos en una base de datos SQL Server permiten agrupar los archivos de datos, sea el principal o los secundarios.

Toda base de datos tiene el grupo de archivos llamado **PRIMARY**, que contiene el archivo principal y pueden contener otros archivos de datos definidos por el usuario que son los tipos de archivos secundarios.

Se pueden crear adicionalmente más grupos de archivos con el objeto de agrupar los archivos secundarios. El objetivo de crear más grupos puede ser el de optimizar la base de datos ubicando la información de diferentes módulos de la empresa en diferentes discos, lo que hace más efectivo el acceso a la información.

Los tipos de grupos de archivos se definen de la siguiente manera:

Grupo de archivos	Descripción
PRIMARY	Grupo de archivo que contiene el archivo primario y puede contener más archivos secundarios, todas las tablas del sistema se asignan a este grupo.
Definidos por el usuario	Grupo de archivo creado por el usuario para agrupar archivos secundarios de base de datos. Se pueden crear al crear la base de datos o se pueden crear al modificar la base de dato.

Cuando se crean objetos en la base de datos sin especificar a qué grupo de archivos pertenecen, se asigna al grupo de archivos predeterminado.

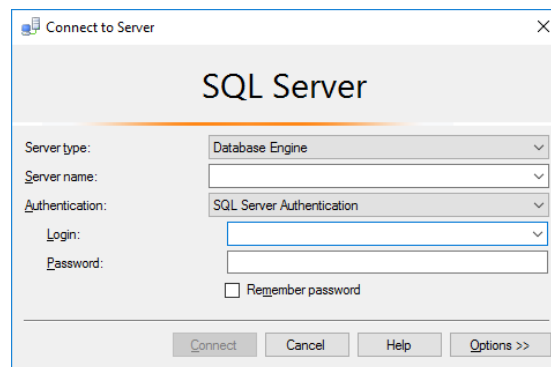
Los archivos del grupo predeterminado deben ser suficientemente grandes como para dar cabida a todos los objetos nuevos no asignados a otros grupos de archivos.

El grupo de archivos **PRIMARY** es el predeterminado, a menos que se cambie con la instrucción **ALTER DATABASE**.

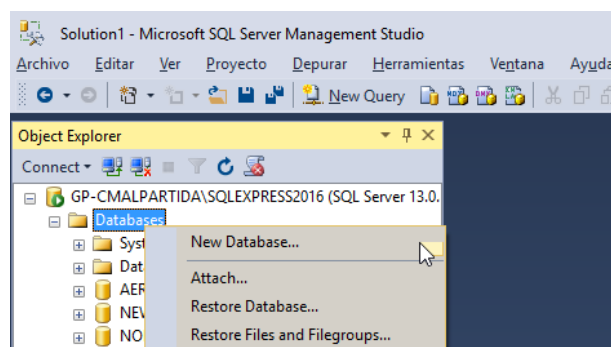
Una vez creada la base de datos, se pueden agregar más grupos de archivos y en estos agregar archivos secundarios para guardar los registros de las tablas.

3.1.2 Creando una base de datos de forma interactiva

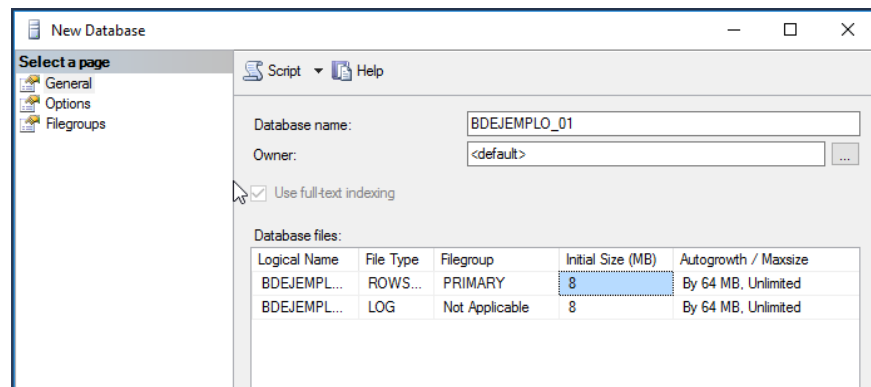
1. Iniciar sesión con SQL Server, confirmando la conexión al servidor cuando se lo pida.



2. Hacer clic derecho en **Base de datos** y elegir la opción **Nueva base de datos**, que se encuentra en el lado derecho de la consola del servidor SQL, tal como se indica a continuación.



3. En el cuadro de diálogo mostrado, ingresar el nombre de la base de datos y pulsar en **Aceptar**.



3.1.3 Creando una base de datos por medio de script

- **Crear una base de datos individual.** En este ejemplo se crea una base de datos llamada **BDEJEMPLO_02** y se especifica un único archivo. El archivo especificado se convierte en el archivo principal y se crea automáticamente, un archivo de registro de transacciones de 1 MB. Como no se especifican MB ni KB en el parámetro **SIZE** del archivo principal, se asigna en megabytes. Ya que no existe **<filespec>** para el archivo de registro de transacciones, este no tiene **MAXSIZE** y puede crecer hasta llenar todo el espacio disponible en el disco.

```
CREATE DATABASE BDEJEMPLO_02
ON
(
    NAME          = BDEJEMPLO_02_DAT,
    FILENAME      = 'C:\dbData\BDEJEMPLO_02_DAT.mdf',
    SIZE          = 5,
    MAXSIZE       = 10,
    FILEGROWTH    = 1
)
go
```

- **Crear una base de datos sin especificar los archivos.** Este ejemplo crea una base de datos llamada **BDEJEMPLO_03** y crea el archivo **principal** y **de registro de transacciones** correspondientes. Debido a que la instrucción no tiene elementos **<filespec>**, el archivo principal tiene el tamaño del archivo principal de la base de datos **model**. Mientras que el registro de transacciones tiene el tamaño del archivo del registro de transacciones de la base de datos **model**.

Como no se ha especificado **MAXSIZE**, los archivos pueden crecer hasta llenar todo el espacio disponible en el disco.

```
CREATE DATABASE BDEJEMPLO_03
go
```

- **Crear una base de datos sin especificar **SIZE**.** Este ejemplo crea una base de datos llamada **BDEJEMPLO_04**. El archivo **BDEJEMPLO_04_DAT** se convierte en el archivo principal, con un tamaño igual al tamaño del archivo principal de la base de datos **model**.

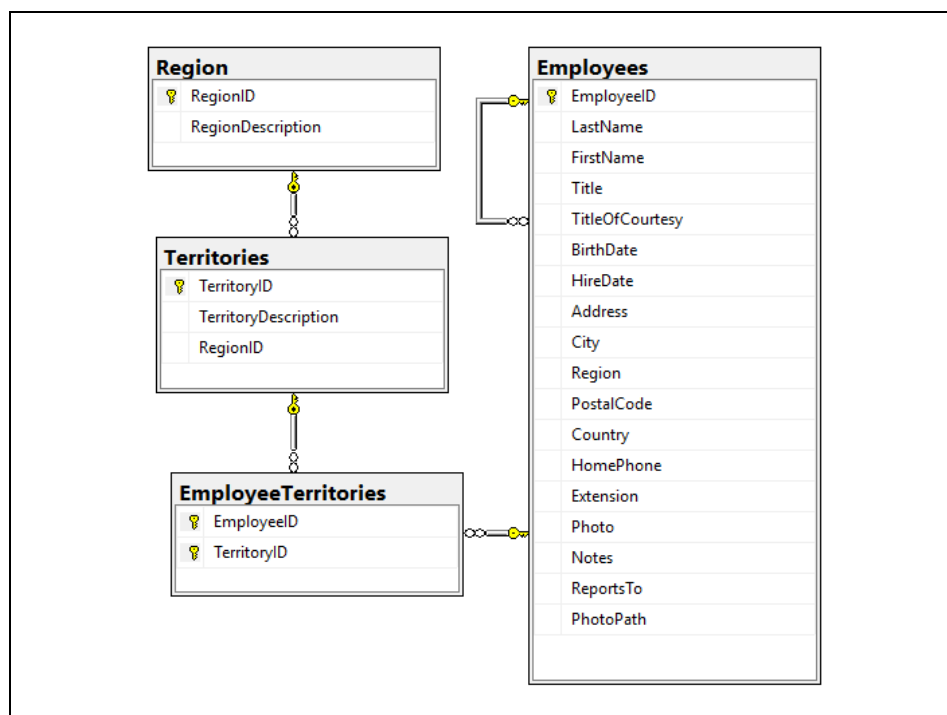
Por su parte, el archivo de registro de transacciones se crea automáticamente y es un 25 por ciento del tamaño del archivo principal o 512 KB, el que sea mayor.

Como no se ha especificado MAXSIZE, los archivos pueden crecer hasta llenar todo el espacio disponible en el disco.

```
CREATE DATABASE BDEJEMPLO_04
ON
( NAME      = BDEJEMPLO_04_DAT,
  FILENAME  = 'C:\dbData\BDEJEMPLO_04_DAT.mdf'
)
go
```

3.2 Creación de las tablas

El DDL provee comandos como CREATE, ALTER y DROP, los cuales permiten la creación, modificación y eliminación de objetos. Si se trata de tablas, se podrán usar las ÓRDENES CREATE TABLE, ALTER TABLE y DROP TABLE.



Por ejemplo, en el siguiente diagrama, se muestra un conjunto de tablas que están ya relacionadas (en base a sus llaves primarias y foráneas).

Además, se puede implementar este diagrama de 2 maneras, tal como se muestra en los siguientes 2 scripts.

3.2.1 Forma directa

Se implementan las tablas, las llaves primarias, las llaves foráneas y las relaciones en forma directa.

```
CREATE DATABASE NORTHWIND
Go

USE NORTHWIND
go
```

```

CREATE TABLE dbo.Region
(
    RegionID          int          NOT NULL,
    RegionDescription char(50)     NOT NULL,
    CONSTRAINT PK_Region PRIMARY KEY CLUSTERED (RegionID)
)
go

CREATE TABLE dbo.Territories
(
    TerritoryID      varchar(20)    NOT NULL,
    TerritoryDescription char(50)    NOT NULL,
    RegionID         int            NOT NULL,
    CONSTRAINT PK_Territories PRIMARY KEY CLUSTERED (TerritoryID),
    CONSTRAINT FK_Territories_Region FOREIGN KEY(RegionID)
        REFERENCES dbo.Region (RegionID)
)
go

CREATE TABLE dbo.Employees
(
    EmployeeID      int IDENTITY (1,1) NOT NULL,
    LastName        varchar(20)        NOT NULL,
    FirstName       varchar(10)        NOT NULL,
    Title           varchar(30)         NULL,
    TitleOfCourtesy varchar(25)         NULL,
    BirthDate       datetime            NULL,
    HireDate        datetime            NULL,
    Address         varchar(60)         NULL,
    City            varchar(15)         NULL,
    Region          varchar(15)         NULL,
    PostalCode      varchar(10)         NULL,
    Country         varchar(15)         NULL,
    HomePhone       varchar(24)         NULL,
    Extension       varchar(4)          NULL,
    Photo           image               NULL,
    Notes           varchar(max)        NULL,
    ReportsTo       int                 NULL,
    PhotoPath       varchar(255)        NULL,
    CONSTRAINT PK_Employees PRIMARY KEY CLUSTERED (EmployeeID),
    CONSTRAINT FK_Employees_Employees FOREIGN KEY(ReportsTo)
        REFERENCES dbo.Employees (EmployeeID)
)
go

CREATE TABLE dbo.EmployeeTerritories
(
    EmployeeID      int            NOT NULL,
    TerritoryID     varchar(20)    NOT NULL,
    CONSTRAINT PK_EmployeeTerritories PRIMARY KEY
        CLUSTERED (EmployeeID, TerritoryID),
    CONSTRAINT FK_EmployeeTerritories_Employees FOREIGN KEY(EmployeeID)
        REFERENCES dbo.Employees (EmployeeID),
    CONSTRAINT FK_EmployeeTerritories_Territories FOREIGN KEY(TerritoryID)
        REFERENCES dbo.Territories (TerritoryID)
)
go

```

3.2.2 Forma paso a paso (empleando ALTER TABLE)

1. Crear las tablas.
2. Agregar las llaves primarias (ADD PRIMARY KEY).
3. Agregar las llaves foráneas y relaciones (ADD FOREIGN KEY – REFERENCES).

```

-- Crear la base de datos
CREATE DATABASE NORTHWIND
go

```

```

-- Activarla con USE
USE NORTHWIND
go

-- Creación de Tablas
CREATE TABLE Region
(
    RegionID          int          NOT NULL,
    RegionDescription char(50)     NOT NULL
)
go

CREATE TABLE dbo.Territories
(
    TerritoryID      varchar(20) NOT NULL,
    TerritoryDescription char(50) NOT NULL,
    RegionID         int         NOT NULL
)
go

CREATE TABLE dbo.Employees
(
    EmployeeID      int IDENTITY (1,1) NOT NULL,
    LastName         varchar(20)       NOT NULL,
    FirstName       varchar(10)        NOT NULL,
    Title           varchar(30)         NULL,
    TitleOfCourtesy varchar(25)        NULL,
    BirthDate       datetime           NULL,
    HireDate        datetime           NULL,
    Address         varchar(60)        NULL,
    City            varchar(15)        NULL,
    Region          varchar(15)        NULL,
    PostalCode      varchar(10)        NULL,
    Country         varchar(15)        NULL,
    HomePhone       varchar(24)        NULL,
    Extension       varchar(4)         NULL,
    Photo           image              NULL,
    Notes           varchar(max)       NULL,
    ReportsTo       int               NULL,
    PhotoPath       varchar(255)      NULL
)
go

CREATE TABLE dbo.EmployeeTerritories
(
    EmployeeID      int          NOT NULL,
    TerritoryID     varchar(20)  NOT NULL
)
go

-- Creación de las llaves primarias
ALTER TABLE dbo.Region
    ADD CONSTRAINT PK_Region PRIMARY KEY
        CLUSTERED (RegionID)
go

ALTER TABLE dbo.Territories
    ADD CONSTRAINT PK_Territories PRIMARY KEY
        CLUSTERED (TerritoryID)
go

ALTER TABLE dbo.Employees
    ADD CONSTRAINT PK_Employees PRIMARY KEY
        CLUSTERED (EmployeeID)
go

ALTER TABLE dbo.EmployeeTerritories
    ADD CONSTRAINT PK_EmployeeTerritories PRIMARY KEY
        CLUSTERED (EmployeeID, TerritoryID)
go

-- Creación de llaves foráneas
ALTER TABLE dbo.Territories
    ADD CONSTRAINT FK_Territories_Region
        FOREIGN KEY (RegionID)
        REFERENCES dbo.Region (RegionID)
go

```

```
ALTER TABLE dbo.EmployeeTerritories
ADD CONSTRAINT FK_EmployeeTerritories_Employees
FOREIGN KEY (EmployeeID)
REFERENCES dbo.Employees (EmployeeID)
go

ALTER TABLE dbo.EmployeeTerritories
ADD CONSTRAINT FK_EmployeeTerritories_Territories
FOREIGN KEY (TerritoryID)
REFERENCES dbo.Territories (TerritoryID)
go
```

3.2.3 Creación de tablas usando datos XML

Es posible incluir en un campo de una tabla datos en formato XML (eXtensible Markup Language), el tipo de datos XML permite almacenar datos que dependen del registro de la misma tabla y posiblemente evitar diseñar un maestro – detalle.

Por ejemplo, en una clínica veterinaria se podría guardar los datos de las mascotas de un determinado propietario en una sola tabla, es decir, Propietarios y Mascotas en la misma tabla.

```
CREATE TABLE dbo.Propietarios
( PropietariosCodigo char(10) NOT NULL,
  PropietariosNombre varchar(200) NOT NULL,
  Mascotas XML NULL,
  PropietariosTelefono varchar(30) NOT NULL,
  CONSTRAINT PK_Propietarios PRIMARY KEY (PropietariosCodigo)
)
go
```

3.2.4 Creación de tablas temporales

Las tablas temporales en SQL Server son utilizadas para almacenar cálculos intermedios en transacciones que requieren de grandes cantidades de datos para ser manejados más eficientemente que con variables. Las tablas temporales se almacenan en la base de datos tempdb.

Las tablas temporales son de dos tipos:

- **Temporales locales** que incluyen en el nombre el símbolo # como primer carácter. Se crean por cada usuario conectado y la tabla se elimina automáticamente cuando el usuario termina la sesión.
- **Temporales globales** que inician con dos símbolos ## en el nombre y son visibles por todos los usuarios conectados al servidor. Al desconectarse todos los usuarios, la tabla temporal global se elimina automáticamente.

Las consideraciones que se deben tener para el trabajo con las tablas temporales son:

- No se pueden usar FOREIGN KEY.
- Las tablas temporales se almacenan en la base de datos del sistema tempdb.
- Una tabla temporal creada en un procedimiento almacenado sólo está presente cuando se completa las transacciones del procedimiento almacenado.

- Las tablas temporales locales se pueden crear con el mismo nombre para diferentes usuarios, SQL Server le incluye un sufijo para diferenciarlas.
- Los dos tipos de tablas se pueden eliminar usando el DROP TABLE.

Ejemplo 01: Creación de una tabla temporal local

```
CREATE TABLE #Prueba
( Codigo          char(04)      NOT NULL,
  Descripcion     varchar(100)  NOT NULL,
  CONSTRAINT PK_Pueba PRIMARY KEY (Codigo)
)
go
```

Ejemplo 02: Creación de una tabla temporal global

```
CREATE TABLE ##DatosEmpresa
( Codigo          char(03)      NOT NULL,
  Descripcion     varchar(100)  NOT NULL,
  Direccion       varchar(100)  NOT NULL
)
go
```

3.3 Uso IDENTITY

IDENTITY es una propiedad que permite que un campo en una tabla se incremente de manera automática al insertar los registros en ella.

Para el uso de la propiedad IDENTITY el tipo de dato debe ser INT. Es necesario definir un valor inicial y un valor de incremento. Es importante anotar que IDENTITY no asegura la unicidad de valor, esta únicamente es posible con la restricción del PRIMARY KEY, UNIQUE o con el INDEX UNIQUE. Solamente, puede existir una columna por tabla con la propiedad IDENTITY.

En el ejemplo, se crea una tabla.

```
CREATE TABLE dbo.Personas
( Codigo          int IDENTITY (1,1) NOT NULL,
  NombreCompleto  varchar(200)     NOT NULL,
  CONSTRAINT PK_Personas PRIMARY KEY (Codigo)
)
go
```

3.4 Tipos de datos definidos por el usuario

SQL Server permite crear tipos de datos que el usuario puede definir en base a los tipos de datos de SQL Server.

La creación de tipos de datos definidos por el usuario va a permitir el uso de los datos nativos de SQL Server de manera mas fácil, por ejemplo, para datos de tipo carácter que tengan una longitud entre 2 a 80 caracteres, se pueden definir un tipo de datos con una longitud de 100 para que almacene cualquier texto que pase los 80 caracteres y ponerle un nombre de fácil recuerdo como Texto100.

3.4.1 Crear tipos de datos definidos por el usuario

Los tipos de datos creados podrán ser utilizados en la base de datos que se crean.

Sintaxis:

```
CREATE TYPE NombreTipoDato FROM TipoDatoSQLServer
```

Ejemplos:

```
CREATE TYPE Codigo10 FROM char(10) NOT NULL
go
CREATE TYPE TextoObligatorio100 FROM varchar(10) NOT NULL
go
CREATE TYPE Precio FROM numeric(9,2) NULL
go

CREATE TABLE dbo.Agencias
( Codigo          Codigo10,
  Descripcion     TextoObligatorio100,
  Responsable     varchar(150),
  CONSTRAINT PK_Agencias PRIMARY KEY (Codigo)
)
GO
```

3.4.2 Eliminar tipos de datos definidos por el usuario

Para poder eliminar un tipo de dato definido por el usuario, este no debe estar en uso en ninguna tabla.

Sintaxis:

```
DROP TYPE NombreTipoDato
```

Ejemplos:

```
DROP TYPE Codigo10
go

/* Mens. 3732, Nivel 16, Estado 1, Línea 1
No se puede quitar el tipo 'Codigo10' porque el objeto 'Agencias' hace
referencia a él. Puede que haya otros objetos que hagan referencia a este
tipo. */
```

3.5 Secuencias

Se puede definir una secuencia como un conjunto de valores que parten de un valor inicial, tienen un incremento o decremento, lo que significa que la secuencia puede ser ascendente o descendente y pueden tener un valor final.

SQL Server permite la creación de secuencias que pueden ser utilizadas para la generación de códigos en las tablas. Lo más importante de las secuencias es que no están ligadas a ningún campo en una tabla. Se recomienda usar la opción de Secuencias en lugar de usar la propiedad IDENTITY.

3.5.1 Tipos de datos permitidos en secuencias

El tipo de dato de la secuencia es un dato entero, los tipos de datos permitidos son:

Tipo de Dato	Valores
Tinyint	Rango de 0 a 255
Smallint	Rango -32,768 a 32,767
Int	Rango -2,147,483,648 a 2,147,483,647
Bigint	Rango -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 Este es el tipo de dato por defecto.
Decimal y Numeric	Con una escala de CERO

Un tipo de dato definido por el usuario creado en base a los tipos de datos anteriores.

3.5.2 Secuencia vs. IDENTITY

En SQL Server se debe usar secuencia en lugar de la propiedad IDENTITY en los siguientes casos:

- La aplicación requiere obtener el valor antes de insertar el registro.
- La aplicación requiere compartir series de números entre múltiples tablas o múltiples columnas en las tablas.
- La aplicación requiere reiniciar el valor de la serie con un valor específico. Por ejemplo. Reiniciar una secuencia que fue creada desde 1 hasta 100 con los mismos valores.
- La aplicación requiere valores que son ordenados por otro campo. La instrucción "NEXT VALUE FOR function" puede aplicarse a la cláusula OVER en la función llamada.
- Una aplicación requiere múltiples valores asignados al mismo tiempo. Por ejemplo, la aplicación necesita obtener tres números seguidos al mismo tiempo.

3.5.3 Creación de una secuencia

Sintaxis:

```
CREATE SEQUENCE [Esquema.]NombreDeSecuencia
[AS [TipoEntero | TipoEnteroDefindoPorElUsuario]]
[START WITH <constante>]
[INCREMENT BY <constante>]
[{MINVALUE [<constant>]} | {NO MINVALUE}]
[{MAXVALUE [<constant>]} | {NO MAXVALUE}]
[CYCLE | {NO CYCLE}]
```

Donde:

NombreDeSecuencia: es el nombre de la secuencia a crear.

TipoEntero: tipo de datos entero de SQL Server.

TipoEnteroDefinidoPorElUsuario: tipo de datos definido por el usuario en base a los números enteros de SQL Server.

Start With: define el valor inicial.

Increment By: define el incremento o decremento.

MinValue: especifica el valor mínimo, por defecto es CERO para el tipo tinyint y un valor negativo para el resto de los tipos.

MaxVale: especifica el valor máximo. El valor por defecto está definido de acuerdo con el valor máximo del tipo de dato entero.

Cycle: permite que la secuencia se reinicie cuando llega a su valor mínimo o máximo, dependiendo si es ascendente o descendente.

Ejemplo 01: crear una secuencia con los valores por defecto.

```
CREATE SEQUENCE ValoresPordefecto
```

Ejemplo 02: crear una secuencia llamada EquipoBasquet que inicia en 1 y termina en 12.

```
CREATE SEQUENCE EquipoBasquet
AS int
START WITH 1
INCREMENT BY 1
MINVALUE 1
MAXVALUE 12
CYCLE
```

3.5.4 Modificación de una secuencia

Modifica los argumentos de una secuencia existente. Para cambiar el tipo de dato numérico de una secuencia, esta se debe eliminar y luego volver a crear con el nuevo tipo.

```
ALTER SEQUENCE [Esquema.]NombreDeSecuencia
[RESTART WITH <constante>]
[INCREMENT BY <constante>]
[{MINVALUE [<constant>]} | {NO MINVALUE}]
[{MAXVALUE [<constant>]} | {NO MAXVALUE}]
[CYCLE | {NO CYCLE}]
```

Ejemplo 01: crear una secuencia con valores por defecto y luego modificarla para que su valor inicial sea 10 y se incremente de 5 en 5.

```
CREATE SEQUENCE PruebaCambio
go

ALTER SEQUENCE PruebaCambio
[RESTART WITH 10]
INCREMENT BY 5
go
```

3.5.5 Eliminar una secuencia

Elimina una secuencia de la base de datos.

```
DROP SEQUENCE [Esquema.]NombreDeSecuencia
```

Ejemplo 01: eliminar la secuencia PruebaCambio.

```
DROP SEQUENCE PruebaCambio
go
```

3.6 Sinónimos

Los sinónimos hacen referencia a los objetos de una base de datos de manera más entendible, clara, o simplemente para abreviar los nombres de tal manera que nos faciliten su uso en nuestras consultas.

Los sinónimos se pueden aplicar a los siguientes objetos de base de datos:

- Procedimiento almacenado del ensamblado (CLR)
- Función con valores de tabla del ensamblado (CLR).
- Función escalar del ensamblado (CLR)
- Funciones de agregado del ensamblado (CLR)
- Procedimiento de filtro de replicación
- Procedimiento almacenado extendido
- Función escalar de SQL
- Función SQL con valores de tabla
- Función SQL con valores de tabla insertados
- Procedimientos almacenados de SQL
- Vistas
- Tabla (definida por el usuario) se incluyen tablas temporales locales y globales.

Debemos tomar en cuenta, el objeto el cual usaremos NO es necesario que exista en el momento de la creación del sinónimo, esto debido a que SQL Server comprueba la existencia de los objetos al momento de llamar o ejecutar el sinónimo.

Sintaxis:

```
CREATE SYNONYM [Esquema.]NombreSinonimo
FOR [Esquema.]NombreObjeto
go
```

Por ejemplo, vamos a crear un sinonimo para la tabla [Orders Details]:

```
CREATE SYNONYM dbo.OD
FOR dbo.[Orders Details]
go
```

Ahora vamos a consultar los datos del sinónimo:

```
CREATE SYNONYM dbo.OD
FOR dbo.[Orders Details]
go
```

Para eliminar el sinónimo, lo podemos hacer de la siguiente manera:

```
DROP SYNONYM dbo.OD
go
```

3.7 Columnas calculadas

Una columna calculada es una columna virtual que no está almacenada físicamente en la base de datos.

Las expresiones de columnas calculadas pueden utilizar datos de otras columnas al calcular un valor para la columna a la que pertenecen.

Existen dos limitaciones o restricciones que se deben de tomar en cuenta y son las siguientes:

- Una columna calculada no puede utilizarse como definición de restricción DEFAULT o FOREIGN KEY ni como NOT NULL.
- Una columna calculada no puede ser el destino de una instrucción INSERT o UPDATE.

3.7.1 Para crear una columna calculada al crear una tabla.

```
-- Crear la base de datos
CREATE DATABASE NORTHWIND
go

-- Activarla con USE
USE NORTHWIND
go

CREATE TABLE dbo.OrderDetails
( OrderID int NOT NULL,
  ProductID int NOT NULL,
  UnitPrice money NOT NULL,
  Quantity smallint NOT NULL,
  Total AS (UnitPrice * Quantity)
)
go
```

3.7.2 Para agregar una nueva columna calculada a una tabla existente.

```
ALTER TABLE dbo.OrderDetails ADD Total AS (UnitPrice * Quantity)
go
```

3.8 La integridad referencial

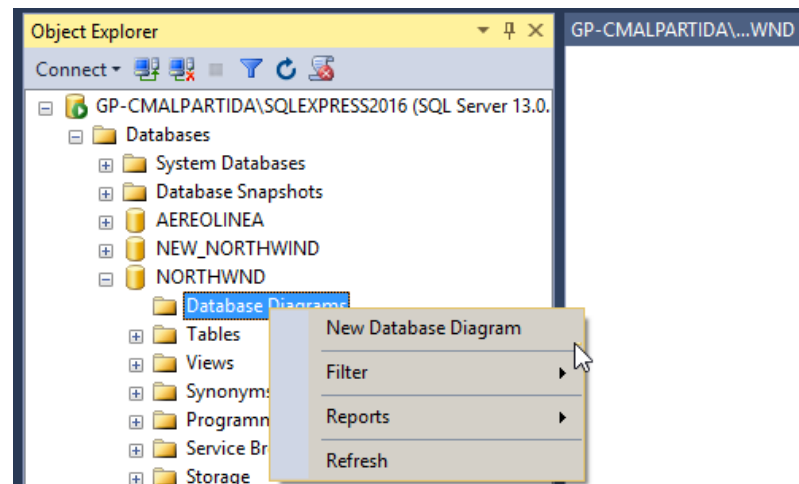
Se puede implementar la integridad referencial al momento de crear las tablas o emplear el comando **ALTER TABLE**. Es importante indicar que las relaciones se deben efectuar con las TABLAS VACÍAS, dado que es posible que si las tablas ya tienen datos, algunos de los registros insertados no cumplan con las reglas de integridad, por lo que las relaciones no se podrían llegar a concretar y en consecuencia, la base de datos no estaría correctamente implementada.

Existen 3 formas de manejar la integridad referencial en caso de actualización de las tablas objetivo (las que contienen la llave primaria):

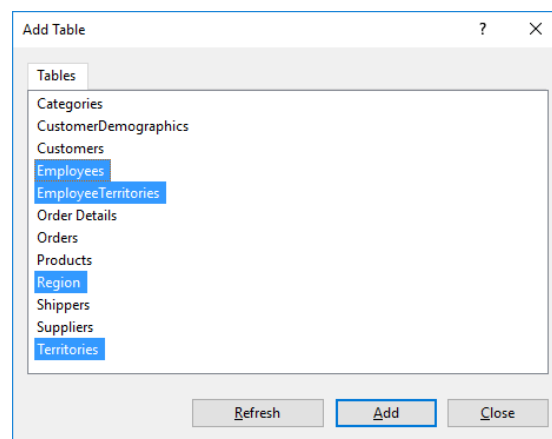
- Restricción (Cancelación)
- Cascada
- Anulación

SQL Server permite crear diagramas con las tablas de una BD, así como, definir las primarias y foráneas.

Por ejemplo, se asume que se ejecutó el script del párrafo anterior donde se crean las tablas de la base de datos **Northwind**. Ahora, se podrá ir al panel izquierdo de la consola de SQL Server, hacer clic en la base de datos **Northwind**, para luego hacer clic derecho en la opción **Diagramas de base de datos** y finalmente, elegir **Nuevo diagrama de base de datos**.

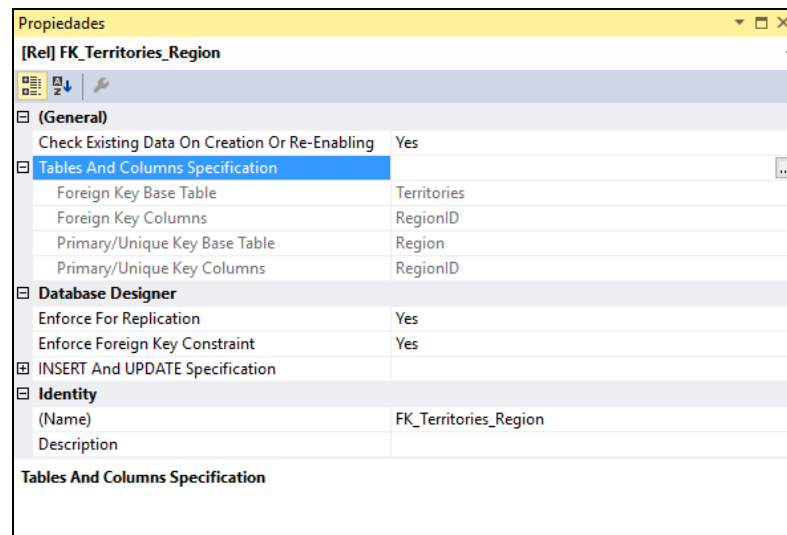


Luego, seleccionar todas las tablas creadas desde el cuadro de diálogo, tal como se aprecia.



Como ya estaban dadas las relaciones, se pueden ver en el diagrama, las tablas con sus **PK** y **FK**, con las líneas de relación entre ellas.

Además, SQL Server establece, por defecto, todas las relaciones bajo el criterio de restricción con lo que respecta a la integridad referencial, tanto para UPDATE como para DELETE. Si se desea, este valor puede cambiarse, bastará con posicionarse en la línea de relación cuya integridad referencial se desea cambiar; desde la ventana de **propiedades** hacer el cambio por cualquiera de las opciones que se muestran.



Aclaración del autor:

Lo que pretende este manual es que el participante se compenetre de mejor forma con el lenguaje SQL, por lo que se obvian algunas formas de trabajo interactivas (es decir, que no requieren el uso de scripts, sino que se pueden hacer directamente, por la consola de SQL Server). De aquí en adelante, se emplearán en la mayoría de casos, scripts para la elaboración de los objetos de la base de datos.

También es importante indicar que al momento que se escriban los scripts, se puede insertar comentarios, empleando los caracteres: /* para abrir comentario y */ para cerrarlos. Además, se puede tener más de una línea de comentario entre estos indicadores. Es necesario documentar sus scripts con comentarios, para un mejor entendimiento de lo que se hace en ellos, así como se recomienda colocar datos descriptivos en la cabecera del script, como el nombre y apellido del creador del script, su fecha de creación, fecha de última modificación, entre otros datos referentes al módulo que se desarrolla.

3.9 Los constraints

Una restricción consiste en la definición de una característica adicional que tiene una columna o una combinación de columnas, suelen ser características como valores no nulos (campo requerido), definición de índice sin duplicados, definición de clave principal y definición de clave foránea (clave ajena o externa, campo que sirve para relacionar dos tablas entre sí).

Se pueden implementar restricciones para permitir que las columnas de las tablas puedan almacenar solo cierto tipo de valores, como, por ejemplo, en la tabla **Products**, la columna **UnitPrice** solo pueda almacenar valores mayores que 0, o que la columna **Discount** en la tabla **OrderDetails**, solo pueda almacenar valores mayores o iguales que 0 y menores o iguales que 1. Este tipo de restricciones se puede implementar con los conocidos constraint de chequeo (**CONSTRAINT CHECK**) y que pueden agregarse en las tablas con la ayuda de **ALTER TABLE**.

Ejemplos:

```
-- Validando el campo BirthDate
ALTER TABLE dbo.Employees WITH CHECK
    ADD CONSTRAINT CK_Birthdate CHECK (BirthDate < getdate())
go
```

```
-- Validando el campo Discount
ALTER TABLE dbo.OrderDetails WITH CHECK
    ADD CONSTRAINT CK_Discount CHECK (Discount >= 0 and Discount <= 1)
go

-- Validando el campo Quantity
ALTER TABLE dbo.OrderDetails WITH CHECK
    ADD CONSTRAINT CK_Quantity CHECK (Quantity > 0)
go
```

**Aclaración del autor:**

Algunos desarrolladores implementan la validación de los datos en las aplicaciones, lo cual es mejor, puesto que a la BD solo debe llegar data validada y coherente. Sin embargo, se pueden implementar restricciones en las tablas, tal como se ha visto en el script anterior, para garantizar la integridad de los datos almacenados, independiente de todos los niveles de validación que se puedan crear en la aplicación.

3.10 Los valores DEFAULT

En SQL un **DEFAULT** es un valor que se asignará a un campo, cuando al insertar un nuevo registro, a dicho campo, no se le asigna valor alguno. Por ejemplo, si se tiene una tabla de **OrderDetails**, con las columnas **OrderID**, **ProductID**, **UnitPrice**, **Quantity**, **Discount**, donde en este último campo, se almacenan valores mayores o iguales que cero; , se puede crear un **DEFAULT** para que si se inserta un nuevo registro, sin necesidad de indicarlo su valor sea 0.

Ejemplo:

```
-- Se establece como default el valor 0 para el campo Discount
ALTER TABLE dbo.OrderDetails
    ADD CONSTRAINT DF_Order_Details_Discount DEFAULT (0) FOR Discount
go

-- Se establece como default el valor 1 para el campo Quantity
ALTER TABLE dbo.OrderDetails
    ADD CONSTRAINT DF_Order_Details_Quantity DEFAULT (1) FOR Quantity
go

-- Se establece como default el valor 0 para el campo UnitPrice
ALTER TABLE dbo.OrderDetails
    ADD CONSTRAINT DF_Order_Details_UnitPrice DEFAULT (0) FOR UnitPrice
go
```

3.11 Creando y manejando índices

Un índice es una estructura que ordena los valores de una o más columnas de una tabla de la base de datos, como la columna **LastName**, **FirstName**, de la tabla **Employees**. Si se busca a una persona en concreto por los campos **LastName**, **FirstName**, el índice ayuda a obtener dicha información con mayor rapidez que si se realiza una búsqueda en todas las filas de la tabla.

El índice proporciona punteros a los valores de datos almacenados en las columnas de la tabla que se seleccionen y, a continuación, clasifica dichos punteros de acuerdo con el orden que se especifique. La base de datos utiliza el índice para buscar un valor en particular.

Como regla general, se debe crear un índice en una tabla solo si los datos de las columnas indexadas se van a consultar con frecuencia. Los índices ocupan espacio en disco y pueden volver lentas las acciones de agregar, eliminar y actualizar filas. En la mayoría de las situaciones, estas desventajas de los índices son compensadas sobradamente por la ventaja que supone recuperar datos a gran velocidad. Sin embargo, si la aplicación actualiza datos con mucha frecuencia o se tienen limitaciones de espacio en disco, puede que sea conveniente limitar el número de índices.

Antes de crear un índice, se deben determinar las columnas que se han de utilizar y el tipo de índice que se desea crear.

Se pueden crear índices tomando como base una sola columna o varias columnas de una tabla de base de datos. Los índices basados en varias columnas permiten distinguir entre filas, en las que una columna puede tener el mismo valor.

Los índices también son útiles si se realizan con frecuencia, procesos de búsqueda o de ordenación basados en dos o más columnas a la vez.

Por ejemplo, si se establecen frecuentemente, criterios para las columnas de nombre y apellido en la misma consulta, conviene crear en estas dos columnas, un índice para varias columnas.

3.11.1 Para determinar la utilidad de un índice

- Se examinan las cláusulas WHERE y JOIN de las consultas. Cada columna incluida en cualquiera de las dos cláusulas, es una posible candidata para un índice.
- Se experimenta con el nuevo índice para observar su efecto en el rendimiento de la ejecución de consultas.
- Se debe tener en cuenta el número de índices que ya se han creado en la tabla. Conviene evitar la existencia de un gran número de índices en una sola tabla.
- Se examinan las definiciones de los índices que ya se han creado en la tabla. Conviene evitar la superposición de índices que contengan columnas compartidas.
- Asimismo, se examina el número de valores de datos únicos en una columna y se compara con el número de filas de la tabla. El resultado es el grado de selectividad de dicha columna, que puede ayudar a decidir si una columna es candidata para un índice y, en caso afirmativo, el tipo de índice apropiado.

3.11.2 Tipos de índices

Dependiendo de la funcionalidad de la base de datos, se pueden crear tres tipos de índices (único, de clave principal y agrupado) en el diseñador de bases de datos.

1. **Índice único.** Es aquel en el que no se permite que dos filas tengan el mismo valor de índice. La mayoría de las bases de datos impiden guardar una tabla con un índice único cuando hay valores de clave duplicados en los datos existentes. La base de datos también puede evitar la incorporación de datos nuevos que creen valores de clave duplicados en la tabla.
2. **Índice de clave principal.** En ocasiones, una tabla de base de datos tiene una columna o una combinación de columnas cuyo valor identifica de forma exclusiva cada fila de la tabla. Esta columna recibe el nombre de clave principal de la tabla.

Si se define una clave principal para una tabla en un diagrama de base de datos, se crea automáticamente un índice de clave principal, que es un tipo concreto de índice único, el cual requiere que cada valor de la clave principal sea único. También permite un acceso rápido a los datos cuando se utiliza dicho índice en consultas.

3. **Índice agrupado.** En un índice agrupado, el orden físico de las filas de la tabla coincide con el orden lógico (indexado) de los valores de clave. Una tabla solo puede contener un índice agrupado.

Mientras que, en un índice no agrupado, el orden físico de las filas de la tabla no coincide con el orden lógico de los valores de clave. Normalmente, un índice agrupado proporciona un acceso a los datos más rápido que un índice no agrupado.



Sugerencia:

Aunque un índice único ayuda a localizar información, se recomienda utilizar en su lugar, restricciones únicas o de clave principal para lograr unos resultados óptimos en cuanto a rendimiento.

3.11.3 ¿Cómo crear un índice?

La sintaxis para crear un índice en una tabla ya definida es la siguiente.

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX índice
ON tabla o vista (campo [ASC|DESC] [, campo [ASC|DESC],...])
[ INCLUDE (campo [...n])]
[ WHERE <filtro>]
[WITH (SORT_IN_TEMPDB = {ON | OFF} | DATA_COMPRESSION= {ON | ROW |
PAGE})]
```

Donde:

Parte	Descripción
índice	Es el nombre del índice a crear.
tabla o vista	Es el nombre de una tabla o vista existente en la que se creará el índice.
campo	Es el nombre del campo o lista de campos que constituyen el índice.
ASC DESC	Indica el orden de los valores de los campos ASC en orden ascendente (valor predeterminado) y DESC un orden descendente.
UNIQUE	Indica que el índice no puede contener valores duplicados.
CLUSTERED	Crea un índice en el que el orden lógico de los valores de clave, determina el orden físico de las filas correspondientes de la tabla. Solo se permite un índice clúster por tabla o vista.

NONCLUSTERED	Crea un índice que especifica la ordenación lógica de una tabla. Cada tabla puede tener hasta 999 índices no clúster.
INCLUDE	Especifica las columnas sin clave que se agregarán en el índice no clúster.
WHERE	Crea un índice filtrado especificando qué filas se van a incluir en el índice.
SORT_IN_TEMPDB	Indica si deben almacenarse resultados temporales de orden en tempdb . El valor predeterminado es OFF.
DATA_COMPRESSION	Especifica la opción de compresión de datos para el índice.

Ejemplos:

```
CREATE NONCLUSTERED INDEX LastName_FirstName  
ON Employees (LastName ASC, FirstName ASC)  
go
```

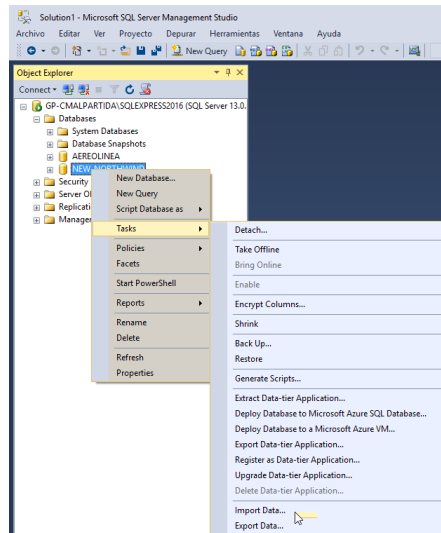
3.11.4 Ventajas y desventajas de los índices

No obstante, las ventajas que ofrecen los índices por lo que respecta al rendimiento también tienen un costo. Las tablas con índices necesitan más espacio de almacenamiento en la base de datos. Asimismo, es posible que los comandos de inserción, actualización o eliminación de datos sean más lentos y precisen más tiempo de proceso para mantener los índices. Cuando se diseña y se crean índices, se deberá tener seguridad de que las ventajas en el rendimiento compensen suficientemente el costo adicional en cuanto a espacio de almacenamiento y recursos de proceso.

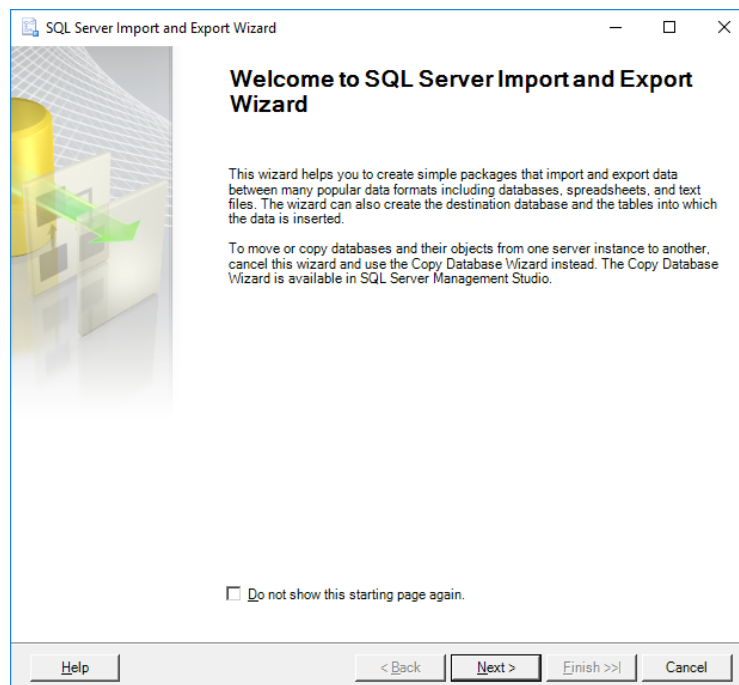
4. Importando datos desde otros orígenes

4.1 Importando datos desde una BD SQL Server

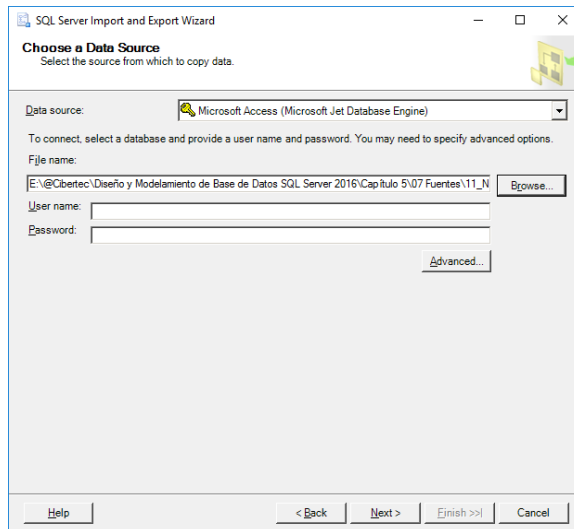
1. Se deberá ubicar la BD desde donde se desea importar los datos. Hacer clic derecho sobre ella y seleccionar **Tasks** (Tareas). Luego, seleccionar **Import Data** (Importar Datos).



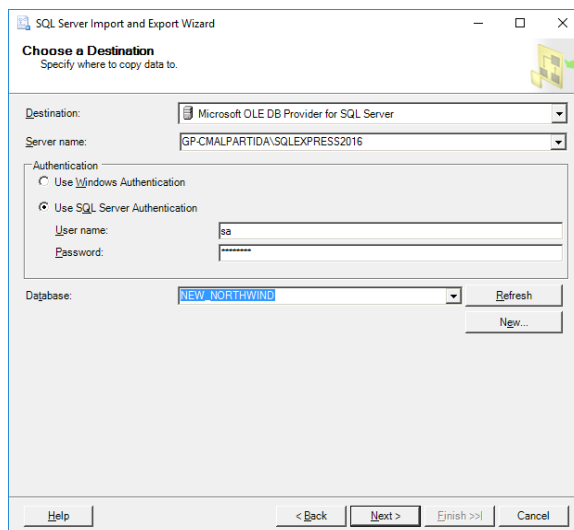
2. Cargar el asistente de importación y exportación SQL Server; para ello, hacer clic en el botón **Next**.



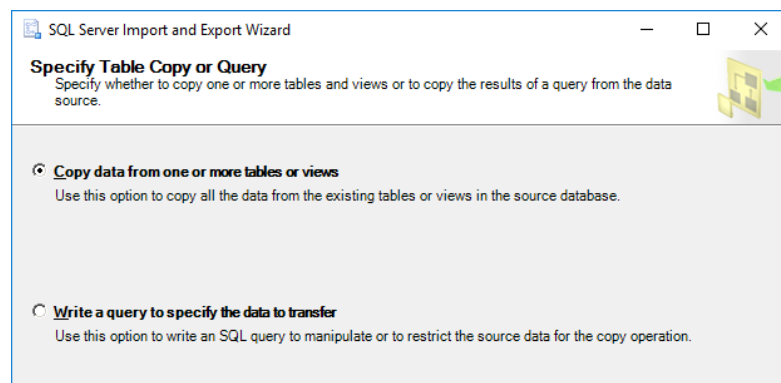
3. Seleccionar la BD de origen.



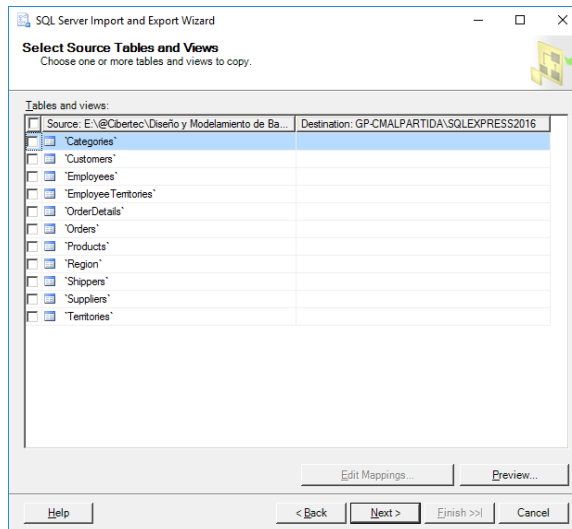
4. Seleccionar la BD destino.



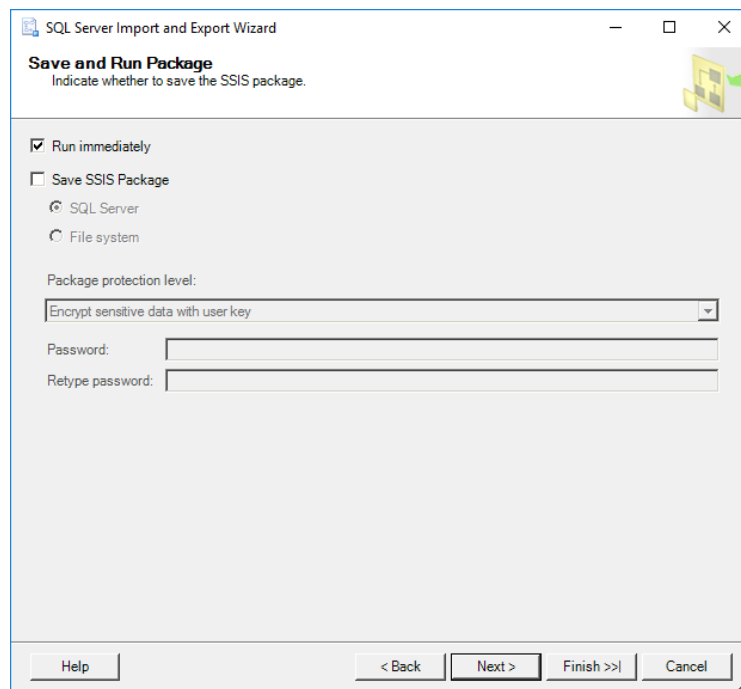
5. Indicar que se desea copiar las tablas y vistas de la BD de origen.



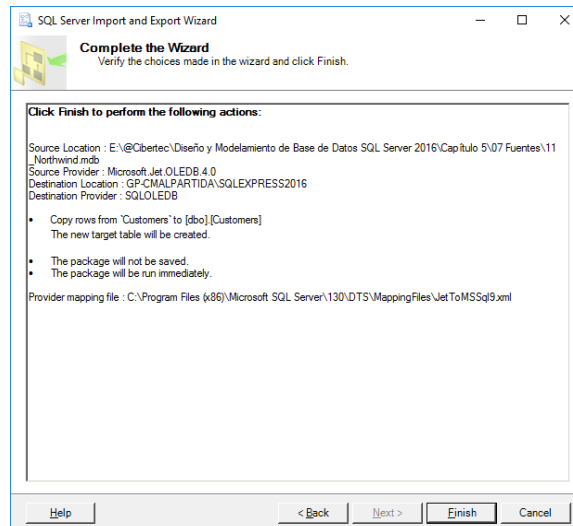
6. Seleccionar ahora los objetos que se desean importar o hacer clic en el botón **select all**, si se desea importar todas las tablas y vistas.



7. Indicar que la importación sea ejecutada de inmediato.



8. Hacer clic en el botón **Finish** para finalizar las especificaciones y proceder a importar lo solicitado.



9. Tras finalizar la importación, hacer clic en **Close**, para cerrar el DTS.

4.2 Importando datos desde una BD MS Access

Se repiten los mismos pasos que en la importación desde SQL Server, pero en el paso 3, se elige como origen de datos una **base de datos MS Access**.

4.3 Importando datos desde Excel

Se repiten los mismos pasos que en la importación desde SQL Server, pero en el paso 3, se elige como origen de datos **un libro MS Excel**.