IES CHAN DO MONTE
C.S. de Desarrollo de Aplicaciones Multiplataforma
Módulo Base de datos

UNIDAD 4 ACTIVIDAD3: Lenguaje de Definición de Datos (DDL) CREACIÓN DE TABLAS

Índice

1. Tipos de datos.

Los **objetos que contienen datos** tienen asociado un **tipo de datos**.

Los tipos de datos especifican el tipo de información (caracteres, números o fechas) que puede contener, así como **la forma en la que se almacenan los datos**.

Los siguientes objetos tienen tipos de datos:

- ✓ **Columnas de tablas** y **vistas**.
- ✓ **Parámetros de procedimientos almacenados y funciones, así como los valores que devuelven**.
- ✓ **Variables**.

Al asignar un **tipo de datos a un objeto** se definen **cuatro atributos del objeto**:

- El **tipo de datos** que contiene el objeto.
- La **longitud** o **tamaño del valor almacenado**.
  - ✓ La longitud de un tipo de datos numérico es el número de bytes utilizados para almacenar el número.
  - ✓ La longitud para una cadena de caracteres o tipo de datos Unicode es el número de caracteres.
  - ✓ La longitud para los tipos de datos binary, varbinary y image es el número de bytes.

  Por ejemplo, un tipo de datos int que puede contener 10 dígitos se almacena en 4 bytes y no acepta coma decimal. El tipo de datos int tiene una precisión de 10, una longitud de 4 y una escala de 0.
- La **precisión del número** (sólo tipos de datos numéricos). Es el **número de dígitos** de un número.
- La **escala del número** (sólo tipos de datos numéricos). Es el **número de dígitos situados a la derecha de la coma** decimal de un número

  Por ejemplo, el número 123,45 tiene una precisión de 5 y una escala de 2.

Microsoft SQL Server proporciona **los siguientes tipos de datos**:

- **Tipos de datos de sistema** que define todos los tipos de datos que pueden utilizarse con SQL Server.
- **Tipos de datos definidos por el usuario**. También se puede definir nuestros propios tipos de datos en Transact-SQL

1.1 Tipos de datos proporcionados por el sistema

SQL Server **proporciona varios tipos de datos diferentes**. Ciertos tipos de datos tienen varios tipos de datos proporcionados asociados. Por ejemplo, podría utilizar los tipos de datos int, decimal o float para almacenar datos numéricos. No obstante, debería **escoger tipos de datos apropiados** para **optimizar el rendimiento** y **conservar espacio** en el disco.

Categoría	Tipo de datos
Numéricos exactos	bit, smallint, tinyint , int, bigint, smallmoney, money, numeric, decimal
Numéricos aproximados	float, real
Fecha y hora	datetime , smalldatetime, date, datetimeoffset, datetime2,time
Cadenas de caracteres	char, varchar, varchar(max), text
Cadenas de caracteres Unicode	nchar, nvarchar, nvarchar(max), ntext
Cadenas binarias	binary, varbinary, varbinary(max), image
Otros tipos de datos	cursor, timestamp, hierarchyid, uniqueidentifier, sql_variant, xml, table
Tipo de datos espaciales	geography, geometry

A partir de Microsoft SQL Server 2005, se introduce el **especificador max**. Este especificador expande la capacidad de almacenamiento de los tipos de datos varchar, nvarchar y varbinary.

varchar(max), nvarchar(max) y varbinary(max) se denominan **tipos de datos de valores grandes**. Puede utilizar los tipos de datos de valores grandes para almacenar hasta 2^31-1 bytes de datos.

Importante: Mejor es utilizar los tipos de datos varchar(max), nvarchar(max) y varbinary(max) en lugar de los tipos de datos text, ntext y image (estos tipos de datos en versiones posteriores pueden que no estén).

1.2 Tipos de datos definidos por el usuario.

Los tipos de datos definidos por el usuario (también llamados tipo de datos de alias) se **basan en los tipos de datos del sistema de SQL Server**.

Los tipos de datos de alias definidos por el usuario **son útiles** cuando **varias tablas de una base de datos utilicen un campo con el mismo tipo, tamaño** y posiblemente las mismas restricciones de validez o valores predeterminados de datos. Estos tipos de datos pueden tener enlazados reglas y valores predeterminados.

Cuando se **crea un tipo de datos de usuario**, se debe **suministrar los parámetros siguientes**:

- **Nombre**
- **Tipo de datos del sistema** en el que se basa el nuevo tipo de datos.
- **Nulabilidad** (si el tipo de datos permite valores NULL).

  Si no se define explícitamente, la nulabilidad, se asignará en función de la configuración de los valores NULL ANSI predeterminados de la base de datos o conexión.

Si se **crea un tipo de alias en la base de datos model**, existirá en todas las nuevas bases de datos creadas posteriormente. Sin embargo, si **el tipo de datos se crea en una base de datos definida** por el usuario, sólo **existirá en esa base de datos.**

1.2.1 Crear un tipo de datos definido por el usuario

Utilizando la instrucción **CREATE TYPE**:

Sintaxis:

```
CREATE TYPE [ Esquema. ] nombreTipodeDatos
{ FROM tipodedatosBase
   [ ( precisión [ , escala ] ) ]
   [ NULL | NOT NULL ]
} [ ; ]
```

Argumentos

- **Esquema**: Es el **nombre del esquema** al que pertenece el tipo de datos de alias o el tipo definido por el usuario.
- **nombreTipodeDatos**: Es el **nombre del tipo de datos definido por el usuario**. Los nombres de tipos deben cumplir las reglas de los identificadores.
- **tipodedatosBase**: Es el **tipo de datos suministrado por SQL Server** en el que se basa el tipo de datos definido por el usuario.

```
USE Empresa_Clase
--Creamos un tipo de datos definido por el usuario
CREATE TYPE CODIGOPOSTAL
FROM CHAR(5) NOT NULL
```

- **precisión**: Para **decimal** o **numeric**, es un entero no negativo que indica el **número máximo de dígitos decimales que se pueden almacenar en total,** a ambos lados del separador decimal.
- **escala**: Para **decimal** o **numeric**, es un entero no negativo que indica el **número máximo de dígitos decimales que se pueden almacenar a la derecha del separador decimal**, y debe ser menor o igual que el valor de precisión.
- **NULL | NOT NULL**: Especifica si el tipo puede **contener un valor NULL**. Si no se especifica, el valor predeterminado es NULL.

1.2.2 OBTENER INFORMACIÓN ACERCA DE LOS TIPOS DE DATOS.

Para **obtener información de los tipos de datos** definidos por el usuario de una base determinada o también del sistema, se puede hacer de varias formas:

- Consultando la **vista de catálogo** **sys.types**.

  Esta vista contiene una fila por cada tipo del sistema y definido por el usuario.

  Si **solo se quiere obtener los tipos definidos por el usuario**, se puede utilizar la columna **is_user_defined** en la cláusula **WHERE** de la instrucción **SELECT**.

  El valor de esta columna es **1 = Tipo definido por el usuario** o **0 = Tipo de datos del sistema de SQL Server**

```
USE EMPRESA_CLASE
SELECT * FROM sys.types
```

Resultados | Mensajes

	name	system_type...	user_type...	schema...	principal...	max_length	precisi...	scale	collation_name	is_nullable	is_user_defined
29	char	175	175	4	NULL	8000	0	0	Modern_Spanish_CI_AS	1	0
30	timestamp	189	189	4	NULL	8	0	0	NULL	0	0
31	nvarchar	231	231	4	NULL	8000	0	0	Modern_Spanish_CI_AS	1	0
32	nchar	239	239	4	NULL	8000	0	0	Modern_Spanish_CI_AS	1	0
33	xml	241	241	4	NULL	-1	0	0	NULL	1	0
34	sysname	231	256	4	NULL	256	0	0	Modern_Spanish_CI_AS	0	0
35	CODIGOPOSTAL	175	257	1	NULL	5	0	0	Modern_Spanish_CI_AS	0	1

- Consultando el **esquema de información** **information_schema.domains**:

  Devuelve una fila por cada tipo de datos de alias definido por el usuario al que puede tener acceso el usuario actual en la base de datos actual.
  Por ejemplo: La siguiente consulta devuelve el nombre de los tipos definidos por el usuario, su tipo base y su longitud máxima en caracteres

```
USE EMPRESA_CLASE
SELECT domain_catalog, domain_schema ,domain_name, data_type, character_maximum_length
FROM information_schema.domains
ORDER BY domain_name
```

Resultados | Mensajes

	domain_catalog	domain_schema	domain_name	data_type	character_maximum_length
1	Empresa_Clase	dbo	CODIGOPOSTAL	char	5

1.2.3 Eliminar un tipo de datos definido por el usuario

Utilizando la instrucción **DROP TYPE**:

Sintaxis:

```
DROP TYPE [ Esquema. ] nombreTipodeDatos [ ; ]
```

La instrucción ***DROP TYPE no se ejecuta si se cumple alguna de las siguientes condicion***es:

- Hay **tablas en la base de datos** que **contienen columnas del tipo de datos definido por el usuario**.

  Se puede obtener información acerca de las columnas de tipo de datos definido por el usuario si se realiza una consulta de las vistas de catálogo ***sys.columns*** o ***sys.column_type_usages*.**

- Hay **columnas calculadas, restricciones CHECK, cuyas definiciones hacen referencia al tipo definido por el usuario.**

  Se puede obtener información acerca de estas referencias consultando la vista de catálogo ***sys.sql_expression_dependencies***

- Hay **funciones, procedimientos almacenados o desencadenadores** creados en la base de datos, y utilizan variables y parámetros de tipo de alias o de tipo definido por el usuario.

  Se puede obtener información acerca de los parámetros de tipo de alias o de tipo definido por el usuario si se realiza una consulta de las vistas de catálogo ***sys.parameters* o *sys.parameter_type_usages***.

Ejemplo: En el siguiente ejemplo se supone que ya se ha creado un tipo denominado CODIGOPOSTAL en la base de datos actual.

```
DROP TYPE CODIGOPOSTAL ;
```

Ejemplo: Antes de crear el tipo de datos, se comprueba la existencia. En el caso de que exista, se borra

```
USE Empresa_Clase
--comprar la existencia del tipo de datos definidos por el sistema, si existe se borra
IF EXISTS (Select * from sys.types where name='codigopostal')
        DROP TYPE codigopostal
GO
--Creamos un tipo de datos definido por el usuario
CREATE TYPE CODIGOPOSTAL
FROM CHAR(5) NOT NULL
```

2. Crear tablas.

Después de haber definido todos los tipos de datos para su tabla, se puede crear tablas, agregar y eliminar columnas y generar valores de columnas.

Cuando se crea una tabla, se debe especificar **el nombre de la tabla**, los **nombres de las columnas** y los **tipos de datos de las columnas**.

Los nombres de las columnas deben ser únicos para una tabla en concreto, pero puede utilizar el mismo nombre de columna en tablas diferentes en la misma base de datos. Debe nombrar un tipo de datos para cada columna. También se pueden especificar otras opciones para las columnas.

2.1 Forma sencilla de crear una tabla

```
CREATE TABLE [nombreBaseDatos ]. [Esquema ] . nombreTabla
(     nombreColumna  tipoDatos  [NULL  | NOT NULL]
[,...n])
```

- **nombreBaseDatos:** Es el **nombre de la base de datos** en que se crea la tabla. Se debe especificar el nombre de una base de datos existente. Si no se especifica, *nombreBaseDatos* se utiliza de manera predeterminada la base de datos actual.
- **Esquema:** Es el **nombre del esquema** al que pertenece la nueva tabla. Si no se especifica, *esquema*, se utiliza esquema predeterminado.
- **nombreTabla:** Es el **nombre de la nueva tabla**. Los nombres de tablas deben seguir las reglas de los identificadores. *nombreTabla* puede contener un **máximo de 128 caracteres**, excepto para los nombres de tablas temporales locales (nombres precedidos de un único signo de **número #**), que no pueden superar los 116 caracteres.
- **nombreColumna**: Es el **nombre de una columna de la tabla**. Los nombres de columna deben seguir las reglas de los identificadores y deben ser únicos en la tabla. Puede contener hasta 128 caracteres.
- **tipoDato**: Especifica el tipo de datos de la columna y el esquema al que pertenece. El tipo de datos puede ser uno de los siguientes:
  - Un **tipo de datos del sistema** de SQL Server.
  - Un **tipo de datos definido por el usuario** basado en un tipo de datos del sistema de SQL Server.

```
USE Ventas
--Creamos un tipo de datos de alias definido por el usuario
CREATE TYPE codigopostal
FROM char(5) not null
GO
CREATE TABLE Tabla1
 (C1 tinyint,
--Creamos una columna del tipo de datos creado anteriormente, que --acepta valores nulos, se ha cambiado la asignación de not null
  C2 codigopostal null);
```

- **NULL | NOT NULL**: En la definición de la tabla se puede especificar **si se permite valores nulos** en cada columna. Si no especifica **NULL** o **NOT NULL**, SQL Server determina si esa columna puede o no aceptar valores nulos basándose en el criterio predeterminado actual para valores nulos en el nivel predeterminado de la sesión o de la base de datos.

  Ejemplo: El siguiente ejemplo crea la tabla Alumno, especificando las columnas de la tabla, un tipo de datos para cada columna y si esa columna admite valores nulos.

```
CREATE TABLE  Alumnos
 (codigo     smallint  NOT NULL,
  nombre   varchar(20)  NOT NULL,
  apellido1 varchar(25)  NOT NULL,
  apellido2  varchar(25)  NULL);
```

2.2 Generar valores de columna

SQL Server le permite **generar automáticamente valores de columna** utilizando:

- La propiedad de **IDENTITY** o
- La **función NEWID** con el tipo de datos **uniqueidentifier**.

2.2.1 Utilizar la propiedad de IDENTITY

Se puede utilizar la **propiedad IDENTITY** para crear columnas (referidas como columnas IDENTITY (de identidad)) que contienen **valores numéricos secuenciales generados por el sistema que identifican cada fila insertada en una tabla**. Una columna de identidad se utiliza a menudo para los **valores clave principales**.

```
CREATE TABLE tabla
(nombre_columna    tipo_datos_numerico    IDENTITY [(valorInicial, incremento)]    [NOT NULL])
```

- valorInicial: Es el **valor que se utiliza para la primera fila** cargada en la tabla.
- incremento: Es el **valor incremental** que se agrega al valor de identidad de la fila cargada anterior.

Si no se especifica estos dos valores, **el valor predeterminado es (1,1).** Se deben especificar los dos valores o ninguno.

Cuando se **crea una columna con la propiedad IDENTIFY**, SQL Server **genera automáticamente un valor de fila** para esa columna, basándose en el valorInicial que será el valor de la primera fila y aumentará este valor en Incremento en las nuevas inserciones.

Hay que considerar los siguientes hechos acerca de la propiedad de IDENTITY y acerca de recuperar los datos de la misma:

- ✓ **Sólo se permite una columna de identidad por tabla**.
- ✓ Una columna de identidad **no permite valores nulos**.
- ✓ Una columna de identidad **debe ser utilizada** con los tipos de datos de **integer (bigint, int, smallint o tinyint), numeric o decimal**. Los tipos de datos numeric y decimal deben ser especificados con una escala de 0 (decimal(p,0) o numeric(p,0)).
- ✓ Las **restricciones DEFAULT no se pueden utilizar** en las columnas de identidad.
- ✓ De forma predeterminada, **no se pueden insertar datos directamente ni actualizarlos en una columna de identidad**

Ejemplo: Se crea la tabla clase con dos columnas: *IDEstudiante* y *nombre*. La propiedad IDENTITY se utiliza para incrementar el valor de la columna IDEstudiante automáticamente para cada fila que se añada a la tabla. El valor inicial se establece en 100 y el valor de incremento es 5. Los valores en la columna serían 100, 105, 110, 115 y así sucesivamente.

```
CREATE TABLE clas3
(IDEstudiante INT IDENTITY (100, 5) NOT NULL,
 nombre VARCHAR(16))
```

2.2.2 Utilizar la función NEWID y el tipo de datos uniqueidentifier

Aunque la propiedad IDENTITY automatiza la numeración de las filas de una tabla, dos tablas distintas, cada una con sus propias columnas de identificadores, pueden generar los mismos valores. Esto es debido a que se garantiza que ***la propiedad IDENTITY es única sólo para la tabla en la que se utiliza.***

Si una aplicación debe generar una **columna de identificadores única para toda la base de datos**, o bien para **todas las bases de datos de todos los equipos conectados en red**, se usa el tipo de datos **uniqueidentifier** junto con la función **NEWID** (Transact-SQL)

- Para insertar un **valor único global**, se crea una definición **DEFAULT** en la columna que usa la función **NEWID**
- El tipo de **datos uniqueidentifier** almacena un número de identificación único como un **valor binario de 16 bytes**. Este tipo de datos se utiliza para almacenar un identificador global único (GUID)

  He aquí un ejemplo de un GUID: 24F1B644-9DD7-11D2-993C-204C4F4F5020
- La función NEWID o NEWSEQUENTIALID crea un identificador único (GUID) que puede ser almacenado utilizando el tipo de datos uniqueidentifer. Nunca se creará el mismo GUID dos veces, así que identifica de manera única una fila a través de tablas, bases de datos, servidores y organizaciones.

Ejemplo: En este ejemplo se crea la columna IdVendedor de la tabla Vendedor con el tipo de datos uniqueidentifier, con un valor predeterminado creado por la función NEWID. Para cada nueva fila en la columna IdVendedor se generará un valor único.

```
CREATE TABLE vendedor
(IdVendedor  uniqueidentifier  NOT NULL DEFAULT NEWID(),
NombreVendedor char(30)  NOT NULL)
```

2.3 Tipos de integridad de datos

La **integridad de datos** se refiere a la **consistencia y exactitud de los datos** almacenados en una base de datos.

Hay **cuatro tipos de integridad de datos**:

- **Integridad de dominio**:

  La integridad de dominio (o columna) especifica **el grupo de valores de datos que son válidos para una columna** y determina si se permiten valores nulos.

- **Integridad de entidad**:

  La integridad de entidad (o tabla) requiere que **todas las filas en una tabla tengan un identificador único**, conocido como el **valor de clave principal**.

- **Integridad referencial**:

  La integridad referencial asegura que la relación entre la clave principal (en una tabla referenciada) y **la clave foránea** (en cada una de las tablas de referencia) **se mantenga siempre**. El mantenimiento de esta relación significa que:

  - **No se puede eliminar una fila en una tabla referenciada**, **ni se puede cambiar la clave principal**, si **una clave foránea hace referencia a la fila**. Por ejemplo, no puede borrar un cliente que ha realizado uno o más pedidos.
  - **No se puede agregar una fila a una tabla de referencia** si la **clave foránea no es igual a la clave principal** de una fila existente en la tabla referenciada. Por ejemplo, no puede crear un pedido para un cliente que no existe.

- **Integridad definida por el usuario**:

  La integridad definida por el usuario permite **definir reglas de negocio específicas** que no se engloban en las otras categorías de integridad. Todas las categorías de integridad soportan integridad definida por el usuario. Hay **restricciones, procedimientos almacenados y desencadenadores** disponibles para aplicar integridad definida por el usuario.

2.3.1 Determinar qué tipo de restricción utilizar

Las **restricciones** son un **método** **ANSI-estándar** de aplicar la **integridad de datos**. Cada tipo de integridad de datos (dominio, entidad y referencial) se aplica con tipos diferentes de restricciones. Las **restricciones** aseguran que se **introducen en las columnas valores de datos válidos y que se mantienen las relaciones entre tablas**.

La tabla siguiente describe los diferentes tipos de restricciones:

Tipo de integridad	Tipo de restricción	Descripción
Dominio	**DEFAULT**	Especifica el **valor que se proporcionará a la columna cuando no se le ha proporcionado un valor explícito** en la instrucción INSERT
	CHECK	Especifica una **regla de validez** para los valores de datos en una columna
	FOREIGN KEY	**Los valores en la(s) columna(s) de clave foránea deben coincidir con los valores** en la(s) columna(s) de **clave principal** de la tabla referenciada.
Entidad	**PRIMARY KEY**	**Identifica cada fila de manera única** (se asegura de que los usuarios no introduzcan valores duplicados **y que se cree un índice para mejorar el rendimiento**). **No se permiten valores nulos**
	UNIQUE	**Evita los duplicados en las claves alternativas y se asegura de que se cree un índice para mejorar el rendimiento**. **Se permiten valores nulos.**
Referencial	**FOREIGN KEY**	Define una columna o combinación de columnas **cuyos valores coinciden con la clave principal de la misma o de otra tabla.**
Definidos por el usuario	**CHECK**	Especifica **una regla de validación** para los valores de los **datos en una columna**

2.3.2 Definir restricciones (CONSTRAINT)

Las restricciones se crean utilizando la instrucción **CREATE TABLE (cuando se crea la tabla)** o **ALTER TABLE(cuando se modifica)**

Se puede **agregar restricciones a una tabla con datos existentes** y se puede establecer **restricciones en una o varias columnas**.

Las restricciones se pueden **definir a nivel de columna o a nivel de tabla**:

- Si la restricción se define **en una única columna,** añadiéndola a su definición, se denomina una restricción a **nivel de columna.**
- También existen **restricciones de tabla**, son restricciones que se definen **después de definir todas las columnas** de la tabla y que pueden **afectar a una o varias columnas de la tabla**.

✓ *Ejemplo de creación de una restricción a **nivel de columna***

```
CREATE TABLE Agenda
(num int NOT NULL CONSTRAINT PK_NUM PRIMARY KEY,
 nombre varchar(20) NOT NULL,
 fechanac smalldatetime NULL CONSTRAINT DF_FECHANACIMIENTO DEFAULT getdate()
                                        --Las restricciones DEFAULT siempre se definen a nivel de columna
)
```

✓ *Ejemplo de creación de una restricción a **nivel de tabla***

```
--Ejemplo de creación de una restricción a nivel de tabla
CREATE TABLE Agenda
(num int,
 nombre varchar (20),
 fechanac smalldatetime CONSTRAINT DF_FECHANACIMIENTO DEFAULT getdate()
 CONSTRAINT PK_NUM PRIMARY KEY(num)   --ahora hay que especificar la columna o columnas afectadas por la restricción
)
```

A la hora **de definir restricciones**, debemos tener en cuenta:

- ✓ La sintaxis para definir una restricción de tabla es muy parecida a la sintaxis de la misma restricción de columna, lo que varía es que en la **restricción a nivel de tabla se tiene indicar las columnas afectadas por la restricción**.
- ✓ Las **restricciones a nivel de tabla tabla se hacen imprescindibles cuando la restricción afecta a varias columnas** o en una restricción **CHECK se hace referencia a otra columna o columnas de la tabla**.
- ✓ Las restricciones **DEFAULT deben ser a nivel de columna**.
- ✓ **No pueden hacer referencia a columnas de otras tablas**.
- ✓ Cuando definimos una restricción **a nivel de columna es opcional poner la palabra clave CONSTRAINT** y el nombre de la restricción. Deberían especificarse **siempre nombres para las restricciones cuando se crean**, ya que SQL Server proporciona nombres complicados generados por el sistema.

Por ejemplo, dada una tabla de productos en la que la clave principal está formada por el código de fabricante y código de producto (porque dos productos diferentes pueden tener el mismo código de producto con proveedores diferentes).

```
--No podemos definir la clave de  esta manera:
  CREATE TABLE Productos (
    Codproducto int PRIMARY KEY,
    Codproveedor int PRIMARY KEY,
    … )
```

SQL Server **entendería que queremos definir dos claves primarias** y eso es **imposible**. En este caso habría que utilizar una **restricción a nivel de tabla**:

```
  --Debemos la  definir la clave a nivel de tabla:
CREATE TABLE Productos (
    Codproducto int,
    Codproveedor int,
    ……..          --las demás columnas,
    CONSTRAINT PK-Clave PRIMARY KEY (Codproducto, Codproveedor)
);
```

2.4 La sintaxis completa del CREATE TABLE

Sintaxis:

```
CREATE TABLE  [nombreBaseDatos] . [Esquema ]. Nombre_Tabla
( { <definición_columna>
  | <restricción_tabla>  }
   [,...n]
)
dónde

<definición_columna> ::= { nombre_columna  tipo_datos [ NULL | NOT NULL ] }
[ [CONSTRAINT nombre_restricción]
{ DEFAULT expresión_constante
| CHECK (expresión_lógica)
| PRIMARY KEY [CLUSTERED | NON CLUSTERED]
| UNIQUE [CLUSTERED | NON CLUSTERED]
| [FOREIGN KEY] REFERENCES ref tabla [(ref_ columna[,...n] )
        [ON DELETE  { NO ACTION | SET NULL | CASCADE | DEFAULT} ]
        [ON UPDATE  { NO ACTION | SET NULL | CASCADE | DEFAULT} ]  [ ...n]
 Y
<restricción_tabla> ::=
[CONSTRAINT nombre_restricción]
{ CHECK (expresión_lógica)
| PRIMARY KEY [CLUSTERED | NON CLUSTERED] (columna[ ASC | DESC ][,...n])
| UNIQUE [CLUSTERED | NON CLUSTERED] (columna[ ASC | DESC ][,... [,...n] )
| FOREIGN KEY (columna[,...n] )
     REFERENCES ref_tabla [(ref_ columna[,...n] )
        [ON DELETE  { NO ACTION | SET NULL | CASCADE | DEFAULT} ]
        [ON UPDATE  { NO ACTION | SET NULL | CASCADE | DEFAULT} ]
```

Ejemplo 1:

Este ejemplo crea una tabla PERSONAL y define restricciones PRIMARY KEY y CHECK a nivel de columna en dos de las columnas de la tabla. La restricción CHECK asegura que el tipo para una persona sólo puede ser "DOCENTE' o 'NO DOCENTE'.

```
CREATE TABLE  Personal
        (codigo  smallint CONSTRAINT PK_PERSONAL PRIMARY KEY,
         nombre  varchar(20) NOT NULL,
         apellido1 varchar(25) NOT NULL,
         apellido2  varchar(25),
         tipo  varchar(10) NOT NULL,
         CONSTRAINT  CK_PERSONAL_TIPO  CHECK  (tipo in  ('Docente','No Docente')));
```

Ejemplo2

Este ejemplo crea una tabla Empleado y define restricciones PRIMARY KEY y CHECK a nivel de tabla. La restricción CHECK asegura que la fecha en que fue contratado el empleado es anterior a la fecha en que dejó de trabajar.

```
CREATE TABLE  Empleado
        (codigo  smallint  NOT NULL,
         nombre  varchar(20) NOT NULL,
         apellido1 varchar(25) NOT NULL,
         apellido2  varchar(25)
         data_inicio  datetime  NOT NULL,
         data_fin  datetime  NULL,
         CONSTRAINT  PK_EMPLEADO PRIMARY KEY(codigo),
         CONSTRAINT  CK_EMPLEADO_DATA  CHECK  (data_inicio<data_fin));
```

2.5 Modificaciones y eliminación de las restricciones: ALTER TABLE

Sintaxis parcial de **ALTER TABLE** para modificar y eliminar restricciones:

```
ALTER TABLE [nombreBaseDatos] . [Esquema ]. Nombre_Tabla
{
   [ WITH {CHECK | NOCHECK } ]
  ADD
  {  <column_definition>   --Para añadir una columna y sus restricciones a nivel de columna
    | <restricción_tabla>    --Para añadir una restricción a nivel de tabla
  } [,...n ]
  | DROP   --Para borrar restriccion o una columna
  {
     [ CONSTRAINT ] constraint_nombre
     | COLUMN column_nombre
  } [ ,...n ]
  | [ WITH { CHECK | NOCHECK } ] CONSTRAINT     --Para borrar desactivar o activar una restricción
     { ALL | nombre_restricción [,...n ] }
}
[ ; ]
```

Ejemplo1: se añade una columna a una tabla llamada Ventas que permite valores nulos:

```
ALTER TABLE  Ventas
        ADD comision money NOT NULL
        CONSTRAINT CK_Ventas_Comision   CHECK (comision >200);
```

Ejemplo 2: Borrar un campo de una tabla. Primero hay que borrar todas las restricciones y luego el campo.

```
ALTER TABLE  Ventas
        DROP CONSTRAINT CK_Ventas_Comision  ;
ALTER TABLE  Ventas
        DROP COLUMN comision  ;
```

2.6 Visualizar información de los tablas, restricciones e índices

La siguiente tabla muestra los **procedimientos almacenados** de sistema que muestran información acerca de las tablas, restricciones e índices.

Procedimiento almacenado de sistema o vista de catálogo	Descripción
sp_help	Si se ejecuta sin argumentos, devuelve información de resumen de los objetos de todo tipo que existen en la base de datos actual.
sp_help ['nombre_objeto']	Si se especifica nombre_objeto, devuelve información del objeto. Los conjuntos de resultados que se devuelven dependen del objeto de base de datos de que se trate. *Ejemplo1: Devuelve información de la tabla empleado: información sobre las columnas, índices y restricciones de la tabla.* USE **Empresa_Clase** EXEC sp_help '**Empleado**' *Ejemplo2: Devuelve información de la tabla restricción de clave primaria* exec sp_help '**PK_Departamento**'
sp_helpconstraint 'tabla'	Devuelve la lista de todos los tipos de restricciones, las columnas en que se han definido y la expresión que define la restricción (sólo para las restricciones DEFAULT y CHECK) en una tabla.
sp_helpindex 'tabla'	Devuelve información acerca de los índices de una tabla o vista en la base de datos actual

sys.objects	Contiene una fila para cada objeto de ámbito de esquema definido por el usuario que se cree en la base de datos (tablas, restricciones primary key, unique, default, foreign key, check, procedimientos, funciones, etc.)
sys.tables	Devuelve una fila por cada tabla de la base de datos actual. actualmente sólo con **sys.objects.type** = U.
sys.foreign_keys	Contiene una fila por objeto que constituye una restricción FOREIGN KEY, con **sys.object.type** = F en la base de datos actual
sys.check_constraints	Contiene una fila por cada objeto que es una restricción CHECK, con **sys.objects.type** = "C" en la base de datos actual.
sys.indexes	Contiene una fila por índice de una tabla, una vista o una función con valores de tabla en la base de datos actual

2.7 Restricciones PRIMARY KEY Y UNIQUE

- Una restricción **PRIMARY KEY:**
  - ✓ Define una **clave principal** en una tabla. El valor en la clave principal identifica de manera única cada fila en la tabla y exigen la **integridad de entidad** de la tabla.
  - ✓ Sólo se puede definir **una restricción PRIMARY KEY por tabla**.
  - ✓ Los **valores de la clave principal deben ser únicos.** No puede haber valores duplicados.
  - ✓ **No se permiten valores nulos**. Las columnas de restricciones PRIMARY KEY que se especifiquen al crear la tabla se convierten implícitamente a NOT NULL.
  - ✓ Cuando se asigna una restricción **PRIMARY KEY**, se **crea automáticamente un índice único** para soportar las restricciones de clave primaria. Este índice también permite un **acceso rápido** a los datos cuando se utiliza la clave principal en las consultas.

    Se puede especificar **si se creará un índice agrupado o sin agrupar** (*agrupado es el valor predeterminado si todavía no existe*).

*Nota: Un índice de SQL es una **estructura** asociada a una tabla o vista que **acelera la obtención de filas de la tabla.** Los índices pueden ser **agrupados** o **no agrupados**. En una tabla **sólo puede haber un índice agrupado por tabla**. En cambio, puede haber **varios índices no agrupados**.*

- ✓ *Para tablas que **tienen un índice agrupado**: Las filas de datos en cada página de datos están **almacenadas físicamente en orden** basándose en la **clave del índice agrupado**.*
- ✓ *Para las tablas que **no tienen un índice agrupado**: Las filas de datos **no se encuentran almacenadas en ningún orden** concreto y a medida que se van añadiendo filas se van llenando las páginas y no importa el orden en que se guarden las páginas de datos.*
- ✓ ***Un índice no agrupado** es similar al índice de un libro de texto. Los datos se almacenan en un lugar, el índice en otro, con punteros que señalan hacia la ubicación de almacenamiento de los datos.*

- Una restricción **UNIQUE:**
  - ✓ Sirve para definir las **claves alternativas** de una tabla
  - ✓ **Se puede establecer varias restricciones UNIQUE** en una tabla
  - ✓ Asegura que una columna o grupo de columnas **no permitirán valores repetidos**, es decir, se impone la unicidad de los valores de una columna o el conjunto de columnas. Esta restricción aplica la integridad de entidad.
  - ✓ La restricción UNIQUE **utiliza por defecto un índice único no agrupado**, a menos que se especifique un índice
  - ✓ **Puede permitir valores nulos.**

Ejemplo 1: muestra la definición de columna num de una restricción PRIMARY KEY (por defecto se crea automáticamente un índice agrupado y un nombre a la restricción PRIMARY KEY)

```
CREATE TABLE Agenda
(num int PRIMARY KEY,
 nombre varchar(20),
 fechanac smalldatetime DEFAULT getdate()
)
```

*Como alternativa, se **puede asignar un nombre a la restricción PRIMARY KEY** añadiendo la palabra clave **CONSTRAINT**.*

*Ejemplo 2: En la columna **num** crea una restricción **PRIMARY KEY**, con el nombre PK_NUM, y crea un **índice agrupado** para esta restricción (por defecto, crea automáticamente este índice con la restricción PRIMAY KEY, con lo cual se puede omitir la CLUSTERED) y **no admite valores nulos**. En la columna nombre se crea una restricción **UNIQUE** con el nombre PK_NOMBRE y se crea **un índice no agrupado** (por defecto, crea automáticamente este índice con la restricción UNIQUE, con lo cual se puede omitir la palabra NONCLUSTERED) y **admite valores nulos**.*

```
CREATE TABLE Agenda
(num int CONSTRAINT PK_NUM PRIMARY KEY CLUSTERED,
 nombre varchar(20) CONSTRAINT PK_NOMBRE UNIQUE NONCLUSTERED,
 fechanac smalldatetime DEFAULT getdate(),
 )
GO
```

Ejemplo 3: Muestra una restricción UNIQUE creada en las columnas nombreAlamacen y ciudad de la tabla ALMACENES, donde idAlmacen es actualmente la restricción PRIMARY KEY; no debe haber dos almacenes iguales en la misma ciudad.

```
CREATE TABLE Almacenes
(idAlmacen int ,
 nombreAlmacen varchar(20),
 ciudad var varchar(25),
 CONSTRAINT PK_IDALMACEN PRIMARY KEY   (idAlmacen),
 CONSTRAINT PK_AlmacenCiudad UNIQUE (nombreAlmacen,ciudad)  )
```

- Para **añadir un restricción PRIMARY KEY o UNIQUE**, se usa el comando **ALTER TABLE**. Sólo se puede **añadir restricciones** a una tabla si los **datos que hay en ella cumplen** los requisitos de la restricción.

```
ALTER TABLE [nombreBaseDatos] . [Esquema ]. Nombre_Tabla
ADD CONSTRAINT nombreRestriccion {PRIMARY KEY | UNIQUE}
[CLUSTERED | NONCLUSTERED ] column [ ASC | DESC ] [,...n ] )
```

Ejemplo 4:

```
CREATE TABLE Agenda
(num int not null,
 nombre varchar(20),
 fechanac smalldatetime DEFAULT getdate(),
 )
GO
ALTER TABLE Agenda
ADD CONSTRAINT PK_NUM PRIMARY KEY CLUSTERED (num)

ALTER TABLE Agenda
ADD CONSTRAINT PK_NOMBRE UNIQUE (nombre)
```

- Para **eliminar una restricción PRIMARY KEY o UNIQUE** se utiliza el comando **ALTER TABLE** con la sentencia **DROP CONSTRAINT**.

Ejemplo: 5

```
ALTER TABLE Agenda
DROP CONSTRAINT PK_NUM

ALTER TABLE Agenda
DROP CONSTRAINT PK_NOMBRE
```

2.8 Restricciones FOREIGN KEY

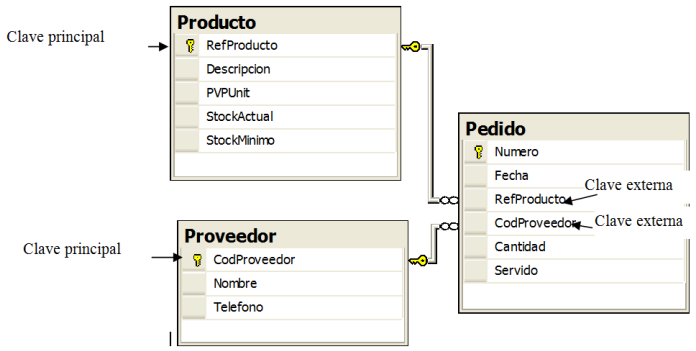
Una restricción **FOREIGN KEY** aplica la **integridad referencial.** La restricción FOREIGN KEY **define una clave foránea(externa), es decir, una referencia a la columna de una restricción PRIMARY KEY de la misma o de otra tabl**a.

Una clave externa (FK) es una columna o combinación de columnas que se utiliza para establecer y exigir un vínculo entre

los datos de dos tablas. En una referencia de clave foránea, se crea un vínculo entre dos tablas cuando las columnas de una de ellas hacen referencia a las columnas de la otra que contienen el valor de clave principal. Esta columna se convierte en una clave foránea para la segunda tabla

Se puede **crear una clave foránea (externa)** mediante la definición de **una restricción FOREIGN KEY** cuando **se crea (CREATE TABLE) o modifica una tabla(ALTER TABLE)**.

En el siguiente ejemplo la tabla **Pedido** *tiene un vínculo a la tabla* **Producto** *porque existe una relación lógica entre Pedido y Producto. La columna* **RefProducto** *de la tabla* **Pedido** *coincide con la* **columna de clave principal** *referencia de la tabla Producto. La columna* **RefProducto** *de la tabla* **Pedido** *es la clave foránea para la tabla* **Producto**. *Lo mismo ocurre con el campo* **CodProveedor** *de la tabla* **Pedido** *que es* **una clave foránea** *para la tabla* **Proveedor**.



Hay que tener en cuenta cuando se define una restricción **FOREIGN KEY**:

- ✓ El **número de columnas y tipos de datos** que se especifican en la instrucción **FOREIGN KEY debe coincidir** con el **número de columnas y tipos de datos** en la cláusula **REFERENCES (con la clave primaria)**. Los *nombres de las columnas no necesitan coincidir*.
- ✓ **puede contener valores NULL**.
- ✓ **Requieren que cada valor de la columna de clave foránea exista en la tabla de referencia**.
- ✓ Solo **hacer referencia sólo a columnas** que sean restricciones **PRIMARY KEY** o **UNIQUE** en la tabla de referencia.
- ✓ Al contrario que las restricciones PRIMARY KEY o UNIQUE, **las restricciones FOREIGN KEY no crean índices automáticamente**. No obstante, una FOREIGN KEY es un buen candidato para indizar, y debería considerar agregar un índice en la(s) columna(s) FOREIGN KEY para mejorar el rendimiento conjunto.
- ✓ Se puede utilizar sólo la cláusula **REFERENCES sin la cláusula FOREIGN KEY** cuando haga referencia a **una columna en la misma tabla.**
- ✓ **Sólo pueden hacer referencia a las tablas de la misma base de datos** en el **mismo servidor**. La integridad referencial entre bases de datos debe implementarse a través de desencadenadores.

Sintaxis de las restricciones FOREIGN KEY de columna:

```
<restricción_columna>::=
[ CONSTRAINT nombre_restriccion ]
 [ FOREIGN KEY ]
    REFERENCES[Esquema.]nombreTablaReferenciada [(columreferenciada ) ]
     [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
     [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
```

Sintaxis de las restricciones FOREIGN KEY a nivel de tabla:

```
< restriccion_tabla> ::=
[ CONSTRAINT nombre_restriccion ]
   | FOREIGN KEY ( column [ ,...n ] )
    REFERENCES nombreTablaReferenciada[(columreferenciada [,...n ]) ]
      [ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
      [ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
```

Ejemplo1: Muestra una restricción FOREIGN KEY de una única columna sobre la tabla Empleado que hace referencia a la tabla Departamento. Sólo se requiere la cláusula REFERENCES para una restricción FOREIGN KEY de una única columna.

```
CREATE TABLE  Empleado
 (codigo   char(10)   PRIMARY KEY,
 Cod_departamento char(3) NOT NULL  REFERENCES DEPARTAMENTO (codigo));
```

*Ejemplo2: También se puede usar explícitamente la cláusula **FOREIGN KEY**. Por ejemplo:*

```
CREATE TABLE  Empleado
  (codigo   char(10)   PRIMARY KEY,
   Cod_departamento char(3) FOREIGN KEY (cod_departamento) REFERENCES DEPARTAMENTO (codigo)) ;
```

Ejemplo3: Vamos a ver un ejemplo de restricción FOREIGN KEY a un atributo de la misma tabla:

```
CREATE TABLE  Empleado
   (codigo   char(10)   CONSTRAINT   PK_EMPLEADO  PRIMARY KEY,
   Cod_jefe char(10) CONSTRAINT  FK_EMPLEADO_JEFE  REFERENCES EMPLEADO (codigo)) ;
```

Ejemplo 4: Las restricciones de claves de varias columnas se crean como restricciones de tabla, al final de las definiciones de las columnas.

```
CREATE TABLE Prestamos(
   NumSocio int NOT NULL,
   CodLibro char(6) NOT NULL,
  NumeroEjemplar smallint NOT NULL,
  FechaPrestamo datetime NOT NULL,
  FechaDevolucion datetime NULL,
--Clave primaria compuesta por varios campos. Hay que definirla a nivel de tabla.
   CONSTRAINT PK_Prestamos1 PRIMARY KEY  (NumSocio, CodLibro,NumeroEjemplar, FechaPrestamo),
--Clave foránea compuesta por varios campos que referencia a la tabla  Ejemplares. Hay que definirla a nivel de tabla.
  CONSTRAINT FK_CodLocEjemplar1 FOREIGN KEY(CodLibro, NumeroEjemplar)
                                REFERENCES Ejemplares (CodLibro, NumeroEjemplar),
--Clave foránea compuesta solo por un campo se puede definir tanto a nivel de columna como de tabla.
  CONSTRAINT FK_SocioPrestamo1 FOREIGN KEY(NumSocio)  REFERENCES Socios (NumSocio)
)
```

- **Restricciones de integridad referencial en cascada**:

Permiten **definir las acciones que SQL Server** lleva a cabo cuando un usuario intenta **eliminar o actualizar una clave** a la que apuntan las **claves foráneas existentes**.

Las cláusulas **REFERENCES** de las instrucciones **CREATE TABLE** y **ALTER TABLE** admiten las cláusulas **ON DELETE** y **ON UPDATE.**

```
[ ON DELETE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
[ ON UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT } ]
```

- **NO ACTION**. Valor por defecto de las opciones. Se **genera un error** y se **revierte la acción de actualización o borrado** de la fila de la tabla primaria.
- **ON DELETE CASCADE. Borra todos los registros cuya clave secundaria es igual que la clave del registro borrado**. Por ejemplo, con esta opción, la eliminación de una fila de datos en la tabla pedidos, provoca la eliminación de todas las filas de datos relacionadas de la tabla Lineas_Pedido.
- **ON UPDATE CASCADE**. Hace que SQL Server **actualice automáticamente los valores contenidos en las columnas relacionadas**, cuando se **actualiza el valor de clave principal** al que hacen referencia.
- **ON DELETE SET NULL** . **Coloca NULL** en todas las **claves secundarias relacionadas** con **la borrada**.
- **ON UPDATE SET NULL.** Coloca **NULL en todas las claves secundarias relacionadas** con **la actualizada**.
- **ON DELETE DEFAULT.** Cuando se **elimina la fila que corresponde a la clave principal** en la tabla referenciada, la clave foránea toma el **valor por defecto definido para esa columna**.
- **ON UPDATE DEFAULT.** Cuando **se actualiza la fila que corresponde a la clave principal** en la tabla referenciada, la **clave foránea** toma el **valor por defecto definido para esa columna**.

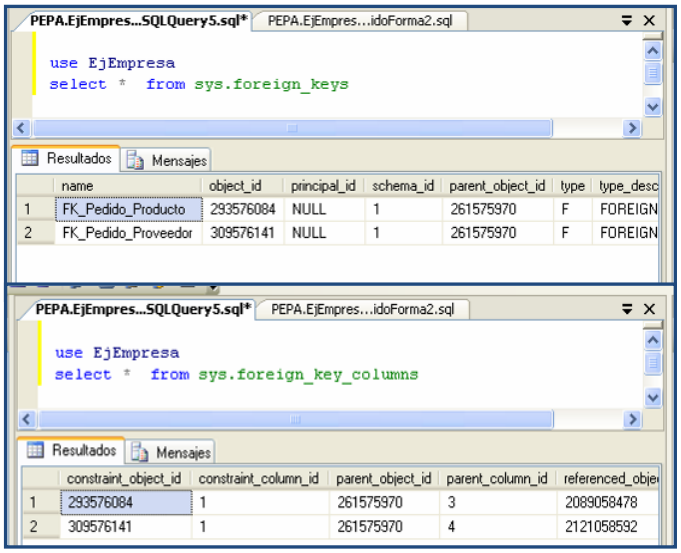
Vamos a ver un ejemplo:

```
CREATE TABLE alquiler
 (dni CHAR(9),
  cod_pelicula CHAR(7),
  CONSTRAINT PK_Alquiler  PRIMARY KEY (dni,cod_pelicula),
  CONSTRAINT FK_ALQUILER_CLIENTES FOREIGN KEY (dni)
                          REFERENCES clientes(dni)   ON DELETE SET NULL,
   CONSTRAINT FK_ALQUILER_PELICULA FOREIGN KEY (cod_pelicula)
                          REFERENCES peliculas(codigo)       ON DELETE CASCADE;
```

- Las siguientes ***vistas de catálogo*** se utilizan para obtener información de las claves foráneas:

✓ Para obtener información acerca de las restricciones FOREIGN KEY: ***sys.foreign_keys***: contiene una fila por objeto que constituye una restricción FOREIGN KEY, con sys.object.type = F.

✓ Para obtener información acerca de las columnas que forman una restricción FOREIGN KEY: ***sys.foreign_key_columns:***: Contiene una fila para cada columna o conjunto de columnas que componen una clave externa.



2.9 Restricciones DEFAULT

Una restricción **DEFAULT** introduce un valor en la columna cuando ésta no ha sido especificada en una instrucción INSERT. Las restricciones DEFAULT aplican la integridad de dominio.

Se puede **definir un valor predeterminado** para una columna usando las siguientes sentencias: **CREATE TABLE**, **ALTER TABLE**

Si una columna **no permite valores NULL y no tiene una definición DEFAULT**, se deberá especificar **explícitamente un valor para la columna** o SQL Server devolverá un error para indicar que la columna no permite valores NULL.

El valor insertado en una columna que se define mediante la combinación de la definición DEFAULT y la aceptación de valores NULL de la columna puede resumirse según se muestra en siguiente la tabla.

Definición de columna	Ninguna entrada, ninguna definición DEFAULT	Ninguna entrada, definición DEFAULT	Si se especifica un valor NULL
Permite valores NULL	NULL	Valor predeterminado	NULL
No permite valores NULL	Error	Valor predeterminado	Error

Debe tenerse en cuenta que:

- ✓ La restricción **DEFAULT** sólo se aplica a **instrucciones INSERT**.
- ✓ Sólo se puede **definir una restricción DEFAULT por columna**.
- ✓ Se pueden aplicar a cualquier columna ***excepto*** a las definidas las que tiene definida la propiedad **IDENTITY** o en columnas con el tipo de datos **timestamp**.
- ✓ **Como valor predeterminado** sólo se puede utilizar **un valor constante**, como por ejemplo, una **cadena de caracteres**, una **función escalar** (función del sistema, definida por el usuario o), **función niládica** o **NULL**.
- ✓ En una definición DEFAULT, en la expresión constante, **no se puede hacer referencia a otra columna de la tabla o** a otras tablas, vistas o procedimientos almacenados.
- ✓ Puede utilizar funciones para proporcionar *valores predeterminados*.

- ✓ El **valor predeterminado debe ser compatible con el tipo de datos de la columna a la que se aplica la definición DEFAULT**. Por ejemplo, el valor predeterminado de una columna int debe ser un número entero, no una cadena de caracteres.

Por ejemplo:

DEFAULT '9952'

DEFAULT getdate() → Función que devuelve la fecha actual.

Ejemplos;

```
CREATE TABLE test_defaults
(keycol smallint not null,
IdProceso smallint DEFAULT @@SPID, -- Id. de sesión del proceso de usuario actual
IdInicioSesion varchar(60) DEFAULT SYSTEM_USER, --predeterminado inicio sesion
IdUsuario varchar(60) DEFAULT USER, --predeterminado usuario actual
fecha datetime DEFAULT getdate(),--predeterminado fecha del sistema
mathcol smallint CONSTRAINT DF_MATHCOL DEFAULT 10 * 2,
char1 char(3),
char2 char(3) DEFAULT 'xyz')
```

- Para **cambiar una definición DEFAULT de una columna o añadir una a una columna**, se puede utilizar el comando **ALTER TABLE.**

- ✓ **Si ya hay definido un valor predeterminado y se quiere cambiar**, lo primero que hay que hacer es **eliminar el valor predeterminado existente** por el nombre y **después añadir el nuevo**.

Para encontrar el nombre que le ha asignado SQL Server al valor predeterminado para poder eliminarlo usando T-SQL, se puede ejecutar el procedimiento almacenado ***sp_help*** con el nombre de la tabla.

```
sp_help test_defaults
go
```

```
go
sp_help test_defaults
go
```

Resultados | Mensajes

	Name	Owner	Type	Created_datetime
1	test_defaults	dbo	user table	2011-01-06 18:44:47.013

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	C
1	keycol	smallint	no	2	5	0	no	(n/a)	(n/a)	N
2	IdProceso	smallint	no	2	5	0	yes	(n/a)	(n/a)	N
3	IdInicioSesion	varchar	no	60			yes	yes	yes	S

	Identity	Seed	Increment	Not For Replication
1	No identity column defined.	NULL	NULL	NULL

	RowGuidCol
1	No rowguidcol column defined.

	Data_located_on_filegroup
1	PRIMARY

	constraint_type	constraint_name	delete_action	update_action	status_enabled
1	DEFAULT on column char2	DF__test_defa__char2__1446FBA6	(n/a)	(n/a)	(n/a)
2	DEFAULT on column fecha	DF__test_defa__fecha__125EB334	(n/a)	(n/a)	(n/a)
3	DEFAULT on column IdInicioSesion	DF__test_defa__IdIni__10766AC2	(n/a)	(n/a)	(n/a)
4	DEFAULT on column IdProceso	DF__test_defa__IdPro__0F824689	(n/a)	(n/a)	(n/a)
5	DEFAULT on column IdUsuario	DF__test_defa__IdUsu__116A8E...	(n/a)	(n/a)	(n/a)
6	DEFAULT on column mathcol	DF__test_defa__mathc__1352D7...	(n/a)	(n/a)	(n/a)

*Vamos a **cambiar la definición DEFAULT del campo char2 de 'xyz' a 'def'**. Hay que recordar que primero tenemos que eliminar el valor predeterminado existente y despues añadir el nuevo. Para eliminar una restricción se utiliza el comando*:

```
ALTER TABLE nombreTabla
DROP CONSTRAINT nombreRestriccion
```

Ejemplo: Eliminar la definición DEFAULT del campo char2

```
ALTER TABLE test_defaults
DROP CONSTRAINT DF__test_defa__char2__1446FBA6  --nombre de la constraint asignada por sql server
```

Ahora se puede añadir el Nuevo valor predeterminado, esta vez dándole un nombre, usando el comando:

```
ALTER TABLE test_defaults
ADD CONSTRAINT DF__test_defaults__char2
DEFAULT 'def' FOR char2
```

*También se puede **añadir una columna completa a una tabla existente y asignarle a dicha columna un valor predeterminado** usando el comando ALTER TABLE como se indica a continuación:*

```
ALTER TABLE nombreTAbla
ADD nombreColumna TipoDeDatos [NULL| NOT NULL]
DEFAULT valor-predeterminado
```

Ejemplo: Vamos a añadir la siguiente columna a la tabla test_default:

```
ALTER TABLE test_defaults
ADD colentera int NULL
DEFAULT 0       --Valor predeterminado
```

- Al **especificar en la nueva columna que admita valores nulos**, a las filas existentes se añadirá el valor **NULL** en la nueva columna.

Ejemplo:

keycol	IdProceso	IdInicioSesion	IdUsuario	fecha	mathcol	char1	char2	colentera
9	58	PEPA\manuel	dbo	06/01/2011 18:12:03	20	NULL	xyz	NULL
11	58	PEPA\manuel	dbo	06/01/2011 18:25:57	20	abc	xyz	NULL
13	58	PEPA\manue	dbo	06/01/2011 19:07:27	20	NULL	def	NULL

- **Si en lugar de NULL, queremos que se asigne el valor por predeterminado de 0 en las filas existentes,** tenemos que utilizar la opción **WITH VALUES** de DEFAULT:

```
ALTER TABLE test_defaults
ADD colentera int NULL
DEFAULT 0 WITH VALUES --Valor predeterminado
```

Ejemplo:

keycol	IdProceso	IdInicioSesion	IdUsuario	fecha	mathcol	char1	char2	colentera
9	58	PEPA\manuel	dbo	06/01/2011 18:12:03	20	NULL	xyz	0
11	58	PEPA\manuel	dbo	06/01/2011 18:25:57	20	abc	xyz	0
13	58	PEPA\manue	dbo	06/01/2011 19:07:27	20	NULL	def	0

- Si **la columna que se añade** se declara como **NOT NULL**, se asigna a todas las filas existentes en la tabla el valor predeterminado 0.

```
ALTER TABLE test_defaults
ADD colentera int NOT NULL
DEFAULT 0    --Valor predeterminado
```

Ej:

keycol	IdProceso	IdInicioSesion	IdUsuario	fecha	mathcol	char1	char2	**colentera**
9	58	PEPA\manuel	dbo	06/01/2011 18:12:03	20	NULL	xyz	**0**
11	58	PEPA\manuel	dbo	06/01/2011 18:25:57	20	abc	xyz	**0**
13	58	PEPA\manue	dbo	06/01/2011 19:07:27	20	NULL	def	**0**

- Cuando añadimos una **columna como NOT NULL a una tabla existente no vacía**, **obligatoriamente tenemos que definir con DEFAULT el valor predeterminado** porque si no se produce el siguiente error:

```
ALTER TABLE test_defaults
ADD colentera int NOT NULL   --se produce un error al no especificar   DEFAULT


Mens. 4901, Nivel 16, Estado 1, Línea 1
ALTER TABLE sólo permite agregar columnas que contengan valores NULL, que tengan la definición DEFAULT, que la columna que se
```

agrega sea una columna de identidad o de marca de hora, o si ninguna de las condiciones anteriores se cumplen, la tabla debe estar vacía para que se pueda agregar esta columna. La columna 'colentera' no se puede agregar a la tabla 'test_defaults' no vacía porque no cumple estas condiciones.

2.10 Restricciones CHECK

Una restricción **CHECK restringe los valores de datos que pueden ser almacenados** en una o más columnas.

La restricción CHECK especifica una **expresión lógica** que debe ser verdad para que los datos sean aceptados. Las restricciones CHECK exigen la **integridad del dominio** mediante la limitación de los valores que puede aceptar una columna.

Por ejemplo, es posible limitar el intervalo de valores para una columna salario creando una restricción CHECK que sólo permita datos entre 400 y 10000 euros. De este modo se impide que se escriban salarios superiores a 10000 euros.

*Para el ejemplo anterior, la expresión lógica sería: **salario>= 400 AND salario<= 10000**.*

Hay que considerar los siguientes hechos cuando aplique una restricción CHECK:

- ✓ **Verifica los datos cada vez que ejecuta una instrucción** INSERT o UPDATE.
- ✓ No se **pueden definir en columnas con la propiedad IDENTITY** o columnas con tipo de datos timestamp, uniqueidentifier, text, ntext o image..
- ✓ No puede **contener subconsultas**.
- ✓ Es posible aplicar varias **restricciones CHECK a una sola columna, en cuyo caso se evalúan en el orden en el que fueron creadas**.
- ✓ También es posible aplicar una sola **restricción CHECK a varias columnas si se crea al nivel de la tabla**.

➢ **Para obtener información acerca de las restricciones CHECK:** se pude consulta la vista de catálogo **sys.check_constraints**. Contiene una fila por cada objeto que es una restricción CHECK, con sys.objects.type = "C".

Sintaxis de las restricciones CHECK de columna:

```
<restricción_columna>::=
        CHECK ( expression_lógica )
}
```

Sintaxis de las restricciones CKECK a nivel de tabla:

```
< restriccion_tabla> ::=
[ CONSTRAINT nombre_restriccion ]
   CHECK   ( expression_lógica )
}
```

Ejemplo:

```
CREATE TABLE Empleados
(       NSS char(12) NOT NULL,
        Nombre varchar(50)  NULL,
        Direccion varchar(50)  NULL,
        Telefono char(9)  NULL,
        Sueldo smallmoney NULL CONSTRAINT CK_Empleados_sueldo  CHECK  (sueldo>(300) AND sueldo <12000),
      CONSTRAINT PK_Empleados1 PRIMARY KEY(NSS),
  )
```

También se puede definir a nivel de tabla. Es imprescindible cuando se hace referencia a otra columna de la tabla

```
CREATE TABLE Empleados
(       NSS char(12) NOT NULL,
        Nombre varchar(50)  NULL,
        Direccion varchar(50)  NULL,
        Telefono char(9)  NULL,
        Sueldo smallmoney NULL,
   CONSTRAINT PK_Empleados1 PRIMARY KEY(NSS),
   CONSTRAINT CK_Empleados_sueldo   CHECK  (sueldo>(300) AND sueldo <12000),
  )
```

Ejemplo 1: Este ejemplo muestra una restricción con nombre con una restricción de modelo sobre los datos de caracteres introducidos

en la columna emp_id

```
CONSTRAINT CK_emp_id CHECK (emp_id LIKE '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][FM]' OR emp_id LIKE '[A-Z]-[A-Z][1-9][0-9][0-9][0-9][0-9][FM]')
```

Ejemplo 2: Este ejemplo especifica que pub_id debe estar en una lista específica o seguir un modelo dado. Esta restricción afecta a la columna pub_id

```
CHECK (pub_id IN ('1389', '0736', '0877', '1622', '1756')  OR pub_id LIKE '99[0-9][0-9]')
```

Ejemplo 3 :

```
CREATE TABLE Trabajo
(
IDTrabajo  smallint IDENTITY(1,1) PRIMARY KEY CLUSTERED,
descripc   varchar(50)  NOT NULL DEFAULT 'desconocido',
valormin tinyint NOT NULL CHECK (valormin>= 10),
valormax tinyint NOT NULL CONSTRAINT CK_valormax CHECK (valormax<= 250)
)
```

Ejemplo 4:

```
CREATE TABLE empleadosTrabajo
(
idEmpl varchar(9) CONSTRAINT PK_emp_id PRIMARY KEY
CONSTRAINT CK_emp_id CHECK (idEmpl    LIKE  '[A-Z][A-Z][A-Z][1-9][0-9][0-9][0-9][0-9][VM]',
/* Cada idEmpleado consiste en tres caracteres, seguido de cinco dígitos en el rango de 10000 a 99999 y por últitmo el sexo del empleado(V o M).*/
nombre   varchar(20)     NOT NULL
)
```

- Para **cambiar los valores de una restricción CHECK**, primero hay que **borrar la restricción** y después **volverla a crearla.**

```
ALTER TABLE trabajo
DROP CONSTRAINT CK_valormax
GO
ALTER TABLE trabajo
ADD CONSTRAINT CK_valormax CHECK (valormax<= 200)
```

2.11 Habilitar la comprobación de datos cuando se agregan restricciones.

Cuando se añade una restricción CHECK o FOREIGN KEY a una tabla existente, las filas que ya existen se comprueban frente a la restricción. Si **no devuelven todas TRUE**, la restricción no se añade a la tabla y SQL Server devuelve un mensaje de error.

Si se desea que las **filas existentes no se comprueben al añadir la restricción CHECK** pero si las actualizaciones o las filas que se añaden posteriormente, se añade la opción **WITH NOCHECK**.

Hay que considerar las siguientes directrices para deshabilitar la comprobación de restricciones en los datos existentes:

- ✓ Se **puede deshabilitar la comprobación** sólo cuando agregue restricciones **CHECK y FOREIGN KEY**.
- ✓ Los datos **siempre se comprueban cuando** se agregan restricciones **PRIMARY KEY y UNIQUE**.
- ✓ Para **deshabilitar la comprobación** de restricciones cuando agrega restricciones **CHECK o FOREIGN** KEY en una tabla con datos existentes, incluya la opción **WITH NOCHECK** en la instrucción **ALTER TABLE**.

Ejemplo: En este ejemplo se agrega una restricción CHECK que verifica que el valor máximo sea menor que 200. Los datos existentes no se comprueban cuando se agrega la restricción.

```
ALTER TABLE trabajo
WITH NOCHECK ADD CONSTRAINT CK_valormax CHECK (valormax<= 200)
```

2.12 Cómo deshabilitar la comprobación de restricciones

La **comprobación de restricciones se puede deshabilitar** para restricciones existentes **CHECK y FOREIGN KEY** existentes para que cualquier dato que se modifique o se introduzca a la tabla no sea comprobado contra estas restricciones.

Para evitar el proceso de comprobación de restricciones, se puede querer deshabilitar restricciones cuando:

- Se está cargando una gran cantidad de datos que sabe que cumplen las restricciones.
- Se quiere cargar una gran cantidad de datos que no cumplen las restricciones. Más adelante se puede ejecutar consultas para cambiar los datos y después rehabilitar las restricciones.

➢ Para **deshabilitar una restricción**, se usa la instrucción **ALTER TABLE** y la cláusula **NOCHECK CONSTRAINT.**

➢ Para **habilitar una restricción deshabilitada**, se usa la instrucción **ALTER TABLE** y la cláusula **CHECK CONSTRAINT.**

Puede **deshabilitar/habilitar todas las restricciones en la tabla** con la instrucción **ALTER TABLE** y con la cláusula **NOCHECK CONSTRAINT ALL** o la cláusula **CHECK CONSTRAINT ALL**.

```
ALTER TABLE trabajo
NOCHECK CONSTRAINT CK_valormax   --deshabilita la restricción
ALTER TABLE trabajo
CHECK CONSTRAINT CK_valormax     --vuelve a habilitar la restricción
```

2.13 Definir una columna calculada.

Sintaxis:

```
column_name AS expression_columna_calculada
[ PERSISTED [ NOT NULL ] ]
```

- expression_columna_calculada:
  - ✓ Es una **expresión** que define el valor de una columna calculada.
  - ✓ Una columna calculada es una **columna virtual no almacenada físicamente** en la tabla y sus valores se vuelven a calcular cada vez que se utilizan en una consulta, a menos que la columna esté marcada con **PERSISTED**.
  - ✓ La columna se calcula a partir de una expresión que utiliza **otras columnas de la misma tabla**.

    *Por ejemplo, una columna calculada podría tener la definición: **cost AS price * qty**.*
  - ✓ La expresión puede ser un nombre de columna no calculada, una constante, una función, una variable o cualquier combinación de estos elementos conectados mediante uno o más operadores.
  - ✓ La expresión **no puede ser una subconsulta** ni contener **tipos de datos de alias**.
- **PERSISTED**: Especifica que SQL Server **almacena físicamente** los valores calculados en la tabla y **actualiza los valores** cuando se actualizan las columnas de las que depende la columna calculada.

Una **columna calculada no puede ser el destino** de una instrucción **INSERT o UPDATE**.

Las columnas calculadas **se pueden utilizar en las listas de selección**, cláusulas WHERE, cláusulas ORDER BY u otras ubicaciones en las que se puedan utilizar expresiones regulares, con las siguientes excepciones:

Las **columnas calculadas** que se utilizan como **restricciones PRIMARY KEY, UNIQUE, CHECK, FOREIGN KEY o NOT NULL** deben marcarse como **PERSISTED**.

```
CREATE TABLE T2
 (C1 INT PRIMARY KEY,
  C2 INT)

CREATE TABLE T3
(C1 INT PRIMARY KEY,
 C2 INT,
 C3 AS C2*5 PERSISTED,
 CONSTRAINT FK_T4 FOREIGN KEY(C3) REFERENCES T2(C1)
 )
```

2.14 Trabajar con esquemas

Los esquemas de base de datos actúan como **espacios de nombres** o contenedores de objetos, como tablas, vistas, procedimientos y funciones dentro de una base de datos.

Se usan para **simplificar la administración** y crean subconjuntos de objetos que pueden administrarse de manera colectiva.

A partir de la versión 2005, se utiliza un nuevo modelo para la administración de las unidades fundamentales de datos dentro de las base de datos.

2.14.1 Creación de esquemas.

El comando de T-SQL para crear esquemas es **CREATE SCHEMA**. Se requiere el *permiso* **CREATE SCHEMA** en la base de datos.

La transacción CREATE SCHEMA también puede crear tablas y vistas dentro del esquema nuevo, y establecer la concesión, denegación o revocación (GRANT, DENY o REVOKE) de permisos sobre esos objetos.

Para crear un objeto especificado dentro de la instrucción CREATE SCHEMA, el usuario debe tener el permiso CREATE correspondiente.

Sintaxis

```
CREATE SCHEMA nombre_esquema
      [ AUTHORIZATION propietario]
```

Argumentos:

- **Nombre_esquema**: Es el **nombre** por el que se identifica al esquema en esta base de datos.
- **AUTHORIZATION propietario**: Especifica el **nombre** de la entidad de seguridad de la base de datos (usuario de base de datos, rol de base de datos –Ej: db_datareader, db_datawriter, db_owner, etc.., o rol de aplicación) que **poseerá el esquema**.

Nota:

La propiedad de los objetos incluidos en el esquema puede transferirse a cualquier entidad de seguridad de base de datos, pero el **propietario del esquema** siempre mantiene el permiso **CONTROL sobre los objetos dentro del esquema**.

Ejemplos:En el ejemplo siguiente se crea el esquema Peliculas, que es propiedad de Luis y contiene la tabla Pelicula.

```
USE Ventas
GO
CREATE SCHEMA Peliculas AUTHORIZATION Luis

CREATE TABLE Películas.Pelicula (codPel int, titulo varchar(30), genero varchar(20))
  GO
```

2.14.2 Ver la información acerca de los esquemas

- **vista de catálogo sys.schemas.:** Contiene una fila por cada esquema de base de datos.

Nombre de columna	Tipo de datos	Descripción
name	sysname	Nombre del esquema. Es único en la base de datos.
schema_id	int	Id. del esquema. Es único en la base de datos.
principal_id	int	Id. de la entidad de seguridad propietaria del esquema.

Ej:

```
use AdventureWorks
select * from sys.schemas
```

Resultados:

name	schema_id	principal_id
dbo	1	1
guest	2	2
INFORMATION_SCHEMA	3	3
sys	4	4
HumanResources	5	1
Person	6	1
Production	7	1
Purchasing	8	1
Sales	9	1

- Vista de esquema de información **INFORMATION_SCHEMA.SCHEMATA**

```
Use AdventureWorks
SELECT * FROM INFORMATION_SCHEMA.SCHEMATA
```

CATALOG_NAME	SCHEMA_NAME	SCHEMA_OWNER	DEFAULT_CHARACTER_SET_CATALOG
AdventureWorks	dbo	dbo	NULL
AdventureWorks	guest	guest	NULL
AdventureWorks	INFORMATION_SCHEMA	INFORMATION_SCHEMA	NULL
AdventureWorks	sys	sys	NULL
AdventureWorks	**HumanResources**	**dbo**	NULL
AdventureWorks	**Person**	**dbo**	NULL
AdventureWorks	**Production**	**dbo**	NULL
AdventureWorks	**Purchasing**	**dbo**	NULL
AdventureWorks	**Sales**	**dbo**	NULL

- **vista de catálogo sys.objects**.

  Contiene una fila para cada objeto de ámbito de esquema definido por el usuario que se cree en la base de datos.

```
use AdventureWorks
select * from sys.objects
```

name	object_id	principal_id	**schema_id**	type	type_desc	create_date
ProductReview	98099390	NULL	**7**	U	USER_TABLE	2011-10-31 17:52:54.650
DF_Address_rowguid	101575400	NULL	**6**	D	DEFAULT_CONSTRAINT	2011-10-31 17:52:54.350
FK_Product_ProductModel_ProductModelID	107147427	NULL	**7**	F	FOREIGN_KEY_CONSTRAINT	2011-10-31 17:53:29.980
TransactionHistory	110623437	NULL	**7**	U	USER_TABLE	2011-10-31 17:52:54.840
DF_ProductReview_ReviewDate	114099447	NULL	**7**	D	DEFAULT_CONSTRAINT	2011-10-31 17:52:54.650
DF_Address_ModifiedDate	117575457	NULL	**6**	D	DEFAULT_CONSTRAINT	2011-10-31 17:52:54.350
FK_Product_ProductSubcategory_ProductSubcategoryID	123147484	NULL	**7**	F	FOREIGN_KEY_CONSTRAINT	2011-10-31 17:53:29.980

2.14.3 Desplazar un objeto a otro esquema

Como ya se comentó, los esquemas son contenedores de objetos y hay ocasiones en que se querrá cambiar un objeto de un contenedor a otro.

Los **objetos pueden desplazarse de un esquema a otro sólo dentro de la misma base de datos**. Al desplazar un objeto a otro esquema, se cambia el espacio de nombre asociado con el objeto, por tanto, se cambia la manera que se accede y consulta el objeto.

Sintaxis

```
ALTER SCHEMA nombre_esquema TRANSFER nombre_objeto
```

Argumentos:

- **nombre_esquema**: Es el nombre **de un esquema de la base de datos actual**, al que se moverá el objeto. **No puede ser SYS ni INFORMATION_SCHEMA**.
- **Nombre_objeto**: Es el nombre un objeto que se va a mover a otro esquema.

ALTER SCHEMA sólo se puede utilizar para **mover objetos entre esquemas de la misma base de datos.**

Para **cambiar o quitar un objeto de un esquema**, se debe utilizar la instrucción ALTER o DROP específica para ese objeto.

Todos los **permisos asociados al objeto se quitarán cuando se mueva el objeto al nuevo esquema**.

Permisos : Para transferir un objeto de un esquema a otro, el usuario actual debe tener el permiso CONTROL para el objeto (no el esquema) y el permiso ALTER para el esquema de destino.

Ejemplos : En el siguiente ejemplo se modifica el esquema HumanResources transfiriendo la tabla Address del esquema Person al esquema.

```
USE AdventureWorks;
GO
ALTER SCHEMA HumanResources TRANSFER Person.Address;
GO
DROP SCHEMA (Transact-SQL).
```

2.14.4 Quitar un esquema de la base de datos.

Cuando ya no se necesite un esquema, se puede eliminar de la base de dato. El esquema que se va **a quitar no puede contener objetos**. Si el esquema contiene objetos, la instrucción DROP registrará errores.

Requiere el **permiso CONTROL** en el esquema o el permiso **ALTER ANY SCHEMA** en la base de datos.

Sintaxis

```
DROP SCHEMA nombre_esquema
```

Argumentos:

- nombre_esquema: Es el nombre por el que se conoce el esquema en la base de datos.

Ejemplos

```
CREATE SCHEMA Peliculas AUTHORIZATION Luis

CREATE TABLE Películas.Pelicula (codPel int, titulo varchar(30), genero varchar(20))
GO
```

Para quitar el esquema, debe quitar primero la tabla contenida en el esquema.

```
DROP TABLE Peliculas.Pelicula;
DROP SCHEMA Peliculas;
GO
```