Consideraciones previas para elegir el tipo de dato asociado a una columna

- Qué tipo de información se va a almacenar. Por ejemplo, no se pueden guardar caracteres en un campo cuyo tipo de datos sea numérico.
- El espacio de almacenamiento necesario (dimensionar el campo).
- Qué tipo de operaciones se van a realizar con los valores del campo. Pues, por ejemplo, no se puede calcular la suma de dos cadenas de texto.
- Si se desea ordenar o indexar por ese campo. Los criterios de ordenación difieren en función del tipo de dato, así, los números almacenados en un campo texto se ordenan según el valor de su código ASCII (1,10,11,2,20,...) que no coincide con la ordenación numérica.

Tipos de Datos (I)

- Numéricos Exactos:

Tipo	Desde	Hasta
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 –1
numeric	-10^38 +1	10^38 –1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Tipos de Datos (II)

Lo marcado en rojo lo incorpora el SQL Server en la versión 2008

- Numéricos Aproximados:

Tipo	Desde	Hasta
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

- Fechas / Horas:

Tipo	Desde	Hasta	Precisión
datetime	1 Enero de 1753	31 Diciembre 9999	3.33 ms.
smalldatetime	1 Enero de 1900	6 Junio 2079	1 minuto
time	N/A	N/A	1 ns.
date	1 Enero de 0001	31 Diciembre 9999	1 día
datetime2	1 Enero de 0001	31 Diciembre 9999	*Variable*
datetimeoffset	1 Enero de 0001	31 Diciembre 9999	*Variable*

Tipos de Datos (III)

- Texto

Tipo	Variable	Unicode	Capacidad
char	NO	NO	8000
varchar	SI	NO	8000
varchar(max)	SI	NO	2^{31}
text	SI	NO	2,147,483,647
nchar	NO	SI	4000
nvarchar	SI	SI	4000
nvarchar(max)	SI	SI	2^{30}
ntext	SI	SI	1,073,741,823

Tipos de Datos (IV)

- Binarios

Tipo	Variable	Capacidad
binary	NO	8000
varbinary	SI	8000
varbinary(max)	SI	2^{31} FILESTREAM *
image	SI	2,147,483,647

* Filestream: posibilidad de almacenar ficheros en Sistema Operativo y controlados por el gestor en vez de en la BD

Tipos de Datos (V)

- Otros tipos de datos

Tipo	Comentario
XML	Almacena una instancia de XML
HierarchyID	Representa la posición en una jerarquía. No representa un árbol.
Table	Un tipo de datos especial que se utiliza para almacenar un conjunto de resultados para un proceso posterior.
Rowversion	Un número único para toda la base de datos que se actualiza cada vez que se actualiza una fila. (utilizar rowversion para versiones futuras)
Sql_variant	Un tipo de datos que almacena valores de varios tipos de datos aceptados en SQL Server, excepto text, ntext, rowversion y sql_variant
Uniqueidentifier	Un identificador exclusivo global (GUID), necesario para replicación
Cursor	Una referencia a un cursor.

Tipos de datos que nos facilita SQL Server y sus características:

bit

Almacena valores boolean, es decir, sólo pueden almacenar valores 1 ó 0, (sin olvidar el valor NULL, que no representa ningún valor). Se utiliza para definir estados: Verdadero/Falso, True/False, on/off, etc...

int

Almacena datos numéricos enteros que pertenezcan al rango de -2^31 (-2.147.483.648) a 2^31-1 (2.147.483.647). Ocupa 4 bytes de memoria en disco y almacena cifras de gran tamaño de funciones matemáticas.

bigint

Este tipo de datos almacena información numérica dentro del rango de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Necesita de 8 bytes para ser almacenado y se utiliza para almacenar aquellas cifras que el tipo de dato int no consigue abarcar.

smallint

Almacena tipos de datos enteros entre el rango -32.768 a 32.767. Necesita de 2 bytes de memoria en disco y se utiliza para aquellas cifras pequeñas que no necesitan de rangos tan superiores como los que permite el tipo de datos int.

tinyint

Permite almacenar datos enteros entre los valores 0 y 255. Sólo requiere de un byte de memoria. Su utilidad se limita a campos cuyo valor numérico sepamos que no supera el valor 255. (edad, piso, número de empleado, etc...)

decimal

Almacena datos con valores de precisión fija y con valores dentro del rango $-10^{38}-1$ a $10^{38}-1$. Utiliza dos parámetros que son precisión y escala.

Por precisión entendemos el número máximo de dígitos decimales capaz de almacenar en el campo. Y por escala el número máximo de dígitos decimales que se almacenará a la derecha del separador decimal. Por ejemplo una precisión de 4 y 3 de escala tiene el siguiente formato: 9,999

numeric

Es lo exactamente el mismo tipo de datos que decimal, pero llamado de otro modo.

money

Utilizado para datos monetarios de -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807, su precisión alcanza la diezmilésima parte de la unidad monetaria representada, necesita 8 bytes de espacio en disco y se utiliza para almacenar sumas de dinero que sean superiores a 214.748,3647.

smallmoney

Almacena datos monetarios de 214.748,3647 a 214.748,3647 con una precisión de la diezmilésima de la unidad monetaria representada. Necesita 4 bytes de espacio de memoria.

float

Permite almacenar datos numéricos de precisión de coma flotante de -1,79E + 38 a 1,79E + 38. Si indicamos el tipo de datos como float(n) siendo n un valor numérico almacenara el valor aproximándolo a n decimales si es que los supera.

real

Es un tipo de dato que almacena valores de coma flotante que van desde -3,40E + 38 a 3,40E + 38. Es otro modo de crear un tipo de dato float(24) ya que real representa 24 números después de la coma decimal.

datetime

Incluye datos de tipo fecha y hora del 1 de enero de 1753 hasta el 31 de Diciembre de 9999. Ocupa 8 bytes de memoria y se utiliza para almacenar fechas y horas muy específicas.

date

Almacena una fecha sin hora. La fecha mínima que permite almacenar es el 1 de enero del año 0001 y como fecha máxima el 31 de diciembre del año 9999 (01-01-0001 a 31-12-9999). Ocupa 3 bytes de almacenamiento con una precisión de 10 dígitos. Este tipo de datos se puede utilizar a partir de la versión 2008.

time

Almacena la hora del día sin la fecha. La hora se almacena con un formato 24 horas, así que la hora mínima que puede almacenar sería 00:00:00.0000000 y la máxima 23:59:59.9999999 reconociendo en estas cadenas las horas, minutos, segundos y las fracciones de segundo. La precisión puede ser de fracciones de segundo y se especifica al crear el tipo de datos. Por defecto la precisión es de 7 dígitos; la precisión es 100ns. Según la precisión varía el espacio de almacenamiento, pudiendo ir desde 3 bytes hasta 2 dígitos; 4 bytes para 3 o 4 dígitos y 5 bytes de 5 a 7 dígitos. Este tipo de datos se puede utilizar a partir de la versión 2008.

datetime2

Es una evolución del tipo datetime original. Permite usar un intervalo de fechas mayor y una precisión fraccionaria de segundos más exacta. La fecha mínima que podríamos almacenar es el 1 de enero del 0001 y la máxima el 31 de diciembre del año 9999. La precisión que se puede indicar es de siete fracciones de segundo. Este tipo de datos se puede utilizar a partir de la versión 2008.

Datetimeoffset

Ofrece reconocimiento de zona horaria. Funciona con la hora local pero además ofrece un último valor que nos indica la diferencia horaria. Un ejemplo del valor devuelto por una columna con este tipo de dato sería:

2009-10-27 08:22:00.1234427 -08:00. Este tipo de datos se puede utilizar a partir de la versión 2008.

smalldatetime

Almacena datos de fecha y hora del 1 de enero de 1900 al 6 de junio de 2079. Necesita 4 bytes de espacio en disco y se utiliza para fechas menos específicas que las almacenadas con datetime.

timestamp

Utilizado para que un registro almacene la hora en el momento en que se inserta y cada vez que es actualizado. Utilizado para trazabilidad y seguimiento de cambios.

uniqueidentifier

La función NEWID() se utiliza para crear identificadores únicos. Se utiliza para crear valores imposibles de aparecer duplicados.

char

Incluye datos de carácter no Unicode de tamaño constante y máximo de 8000 caracteres. Su utilidad se busca en campos cuya longitud será fija como el identificador de una región que siempre tendrá dos letras. Destacar que siempre ocupa la misma cantidad de memoria sin tener en cuenta la longitud de la información almacenada. Si ponemos un campo con tipo char(4) ocupara siempre 4 bytes independientemente de que almacenemos únicamente 2.

varchar

Almacena datos de carácter no Unicode de longitud variable, y con un máximo de 8000 caracteres. Su utilidad se encuentra en campos cuya longitud sea variable, como pueden ser direcciones, nombres, apellidos, etc...

En este caso su tamaño también es variable y depende de la longitud de la cadena almacenada. Esto quiere decir que aunque reservemos 50 caracteres: varchar(50), si luego almacenamos simplemente 10, el tamaño que ocupa corresponde a esos 10 caracteres. Por lo tanto lo que definimos es el tamaño máximo.

varchar(max)

Como tipo de datos coincide con el anterior (varchar), pero al definir como tamaño máximo "max" podemos almacenar una cantidad de bytes de hasta 2^{30}- 1 (2.147.483,67) bytes de datos.

nchar

Permite almacenar datos Unicode de longitud constante y con un máximo de 4000 caracteres. Lo mismo que sucede con todos los tipos de datos Unicode, se utiliza para guardar pequeñas cantidades de texto para usuarios que utilicen diferentes idiomas.

nvarchar

Tipo de datos utilizado para almacenar datos Unicode de longitud variable, y con un máximo de 4000 caracteres. Su modo de almacenar es el mismo que para el tipo de datos nchar, con la diferencia de que el tamaño que ocupa depende del tamaño de los datos almacenados.

nvarchar(max)

Coincide con las características de nvarchar, con la diferencia de que al definir como tamaño (max) puede llegar a almacenar hasta 2^{31} - 1 (2.147.483,67) bytes de datos.

binary

Permite almacenar datos binarios de longitud fija con un máximo de 8000 caracteres. Es interpretado como una cadena de bits, y es utilizado para información dada en base binaria o hexadecimal.

varbinary

Incluye datos binarios de longitud variable y con un máximo de 8000 caracteres. Como sucede con otros tipos de datos var, se diferencia de binary en que su tamaño depende de la cantidad de información almacenada.

varbinary(max)

Tipo de datos con las mismas características que varbinary, pero al definir un tamaño (max), puede llegar a almacenar 2^{31}- 1 (2.147.483,67) bytes de datos. Es el tipo de datos más adecuados para almacenar imágenes y documentos de texto.

NOTA: A partir de SQL Server 2008 se han incluido los tipos de datos varchar(max), nvarchar(max) y varbinary(max), se aconseja aprovechar este tipo de datos y utilizarlos en lugar de text, ntext e image.

xml

Capacitado para almacenar documentes xml.

identity

Identity se define más como una propiedad que como un tipo de datos, su función es incrementar el valor de una columna con cada registro que se inserta. Suele ser utilizado junto con el tipo de datos int, y se utiliza para crear claves primarias auxiliares en muchas ocasiones. Lo veremos más adelante

sql_variant

No es propiamente un tipo de datos. Permite almacenar valores de distintos tipos de datos. Los tipos de datos que no puede llegar a almacenar son:

- varchar(max)
- nvarchar(max)
- text

- image
- sql_variant
- varbinary
- xml
- ntext
- timestamp
- tipos de datos definidos por el usuario

Tipos de dato autonumérico

***IDENTITY** [(semilla , incremento)]*

- ***semilla**:* valor de inicio.
- ***incremento:*** incremento que se aplica

```
CREATE TABLE dbo.herramientas(
   ID INT IDENTITY(1,1) NOT NULL PRIMARY KEY,
   Nombre VARCHAR(40) NOT NULL
)
-- insertamos valores
INSERT INTO dbo.herramientas (Nombre ) VALUES (‘Martillo')
INSERT INTO dbo.herramientas (Nombre ) VALUES (‘Taladro')

-- si borramos, Martillo, se pierde el ID 1. Para reutilizarlo debo establecer

SET IDENTITY_INSERT dbo.herramientas ON
INSERT INTO dbo.herramientas (ID, Nombre) VALUES (1, Serrucho')
```

Lenguaje de definición de datos

- Las instrucciones de definición (IDD) comprenden todas las operaciones necesarias para implantar y mantener un esquema relacional.
- Las IDD permiten crear, modificar y eliminar tablas, así como todos los componentes que las definen: campos, índices, claves, etc. y las restricciones que sean precisas.

EL lenguaje SQL ofrece tres instrucciones generales para la modificación de cualquier objeto.

Instrucción	Descripción
CREATE	Se utiliza para crear un nuevo objeto.
ALTER	Se utiliza para modificar las características de un objeto
DROP	Se utiliza para eliminar objetos.

Lenguaje de definición de datos

- Principales instrucciones:
  - CREATE DATABASE
  - CREATE TABLE
  - ALTER TABLE
  - CREATE VIEW
  - DROP "objeto"

```
CREATE DATABASE miEmpresa

DROP DATABASE miEmpresa
```

Las instrucciones son genéricas para todos los servidores de base de datos, pero cada uno le añade sus propias peculiaridades. Por ejemplo, no es lo mismo una base de datos SQL Server con Windows, con Oracle en Linux, o DB2 bajo OS/400. Entre estos tres entornos varían los directorios de trabajo, por citar un ejemplo.

CREATE DATABASE

No en estándar SQL2003

Para crear una base de datos. Su sintaxis es:

```
CREATE DATABASE nombreBD
[ ON
    [PRIMARY][ < fichero > [ ,...n ] ]
    [ , < grupo_fichero > [ ,...n ] ]
]
[ LOG ON { < fichero > [ ,...n ] } ]
[ COLLATE collation_name ]
[FOR ATTACH |
     FOR ATTACH REBUILT_LOG]

< fichero > ::=
 ( [ NAME = logical_file_name , ]
    FILENAME = 'os_file_name'
    [ , SIZE = size ]

    [ , MAXSIZE = { max_size | UNLIMITED } ]
    [ , FILEGROWTH = growth_increment ] ) [ ,...n ]
< grupo_fichero > ::=
FILEGROUP filegroup_name  [CONTAINS FILESTREAM] < fichero >
    [ ,...n ]
```

- ***nombreBD***: nombre de la BD que se va a crear.
- ***collation_name***: mapa de caracteres
- ***logical_file_name*** : nombre lógico del fichero.
- ***os_file_name*** : nombre físico del fichero.
- ***size***: tamaño del fichero.
- ***max_size***: tamaño máximo del fichero.
- ***growth_increment*** : incremento del fichero.
- ***filegroup_name***: nombre grupo de archivos

Como mínimo, todas las bases de datos de SQL Server tienen dos archivos del sistema operativo: un archivo de datos y un archivo de registro. Los archivos de datos contienen datos y otros objetos, como tablas, índices, procedimientos almacenados y vistas. Los archivos de registro contienen la información necesaria para recuperar todas las transacciones de la base de datos.

Con la cláusula **ON** especificamos los ficheros utilizados para almacenar los archivos de datos.

NAME = N'compras' es el nombre que se utiliza para hacer referencia al archivo en todas las instrucciones Transact-SQL. El nombre de archivo lógico tiene que cumplir las reglas de los identificadores de SQL Server y tiene que ser único entre los nombres de archivos lógicos de la base de datos.

FILENAME = N'C:\data\compras.mdf' es el nombre del archivo físico que incluye la ruta de acceso al directorio. Debe seguir las reglas para nombres de archivos del sistema operativo.

Con la cláusula **SIZE** indicamos el tamaño original del archivo.
Los archivos de SQL Server pueden crecer automáticamente a partir del tamaño originalmente especificado. Con **FILEGROWTH** se puede especificar un incremento de crecimiento y cada vez que se llena el archivo, el tamaño aumentará en la cantidad especificada.

Cada archivo también puede tener un tamaño máximo especificado con **MAXSIZE**. Si no se especifica un tamaño máximo, el archivo puede crecer hasta utilizar todo el espacio disponible en el disco. Esta característica es especialmente útil cuando SQL Server se utiliza como una base de datos incrustada en una aplicación para la que el usuario no dispone fácilmente de acceso a un administrador del sistema. El usuario puede dejar que los archivos crezcan automáticamente cuando sea necesario y evitar así las tareas administrativas de supervisar la cantidad de espacio libre en la base de datos y asignar más espacio manualmente.

Con la cláusula **COLLATE** podemos cambiar la intercalación predeterminada.
La intercalación define:

- El alfabeto o lenguaje cuyas reglas de ordenación se aplican si se especifica la ordenación de diccionario.
- La página de códigos usada para almacenar datos de caracteres que no son Unicode.
- Las reglas de comportamiento frente a mayúsculas y minúsculas y caracteres acentuados.

Modern_Spanish Es un nombre de intercalación de Windows, hay más tipos.
CaseSensitivity: Especifica que sí se distingue entre mayúsculas y minúsculas (CS), o no (CI). *AccentSensitivity:* Especifica si se distinguen los caracteres acentuados (AS), o no (AI). Si no se especifica, se asigna a la base de datos la intercalación predeterminada de la instancia de SQL Server.

CREATE DATABASE: ejemplo

```
CREATE DATABASE [compras]
 ON (NAME = N'compras',
    FILENAME = N'C:\Archivos de programa\Microsoft SQL
    Server\MSSQL11.SQLEXPRESS\MSSQL\DATA\compras.mdf' ,
    SIZE = 2, MAXSIZE = 3000,FILEGROWTH = 10%)
 LOG ON (NAME = N'compras_log',
    FILENAME = N'C:\Archivos de
    programa\Microsoft SQL
    Server\MSSQL11.SQLEXPRESS\MSSQL\
    DATA\compras.LDF' , SIZE = 1,
    FILEGROWTH = 10%)
 COLLATE Modern_Spanish_CI_AS
```

MODIFICAR UNA BASE DE DATOS: ALTER DATABASE. PÁG 110 del libro

BORRADO DE UNA BASE DE DATOS: DROP DATABASE. PÁG 111 del libro

CREAR TABLAS

CREATE TABLE
[database_name . [schema_name] . | schema_name .] **table_name**
({ <column_definition> | <computed_column_definition> |
<column_set_definition>}
[<table_constraint>] [,...n])
[**ON** { partition_scheme_name (partition_column_name) | filegroup | "default" }]
[{ TEXTIMAGE_ON { filegroup | "default" }]
[FILESTREAM_ON { partition_scheme_name | filegroup | "default" }]
[WITH (<table_option> [,...n])] [;]

<column_definition> ::=
column_name <data_type> [FILESTREAM] [COLLATE collation_name]
[NULL | NOT NULL]
[[CONSTRAINT constraint_name] DEFAULT constant_expression]
| [IDENTITY [(seed ,increment)] [NOT FOR REPLICATION]]
[ROWGUIDCOL] [<column_constraint> [...n]]
[SPARSE]

- ***TEXTIMAGE_ON:*** indica en qué grupo de archivos se almacenan las columnas text, ntext e image.
- ***FILESTREAN_ON:*** indica en qué grupo de archivos se almacenan las columnas varbinary
- ***SPARSE: columna con muchas filas NULAS***
- ***FILESTREAM: especifica almacenamiento filestream (fichero SO)***

<table options> ::=
{ DATA_COMPRESSION = { NONE | ROW | PAGE }
[ON PARTITIONS ({ <partition_number_expression> | <range> } [, ...n])] }

<computed_column_definition> ::=
column_name AS c*omputed_column_expression*
[PERSISTED [NOT NULL]]
[[CONSTRAINT *constraint_name*]
{ PRIMARY KEY | UNIQUE }
[CLUSTERED | NONCLUSTERED]
[WITH FILLFACTOR = *fillfactor* | WITH **(** <index_option> [**,** ...*n*]**)**]
| [FOREIGN KEY] REFERENCES *referenced_table_name* [**(** *ref_column* **)**]
[ON DELETE { NO ACTION | CASCADE }]
[ON UPDATE { NO ACTION }]
[NOT FOR REPLICATION]
| CHECK [NOT FOR REPLICATION] **(** *logical_expression* **)**
[ON { *partition_scheme_name* **(** *partition_column_name* **)** | *filegroup* | "default" }]]

Restricción a nivel de columna

```
CONSTRAINT nombre
 { [ NULL | NOT NULL ] |
  [ { PRIMARY KEY | UNIQUE }
       [ CLUSTERED | NONCLUSTERED ]
       [ WITH FILLFACTOR = factor_relleno ]
       [ ON {grupo_ficheros | DEFAULT} ] ]
    ] |
  [ [ FOREIGN KEY ]
       REFERENCES otra_tabla[ (campo_externo1) ]
       [ ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT } ] [
       ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT } ] [
       NOT FOR REPLICATION ]
    ] |
  CHECK [ NOT FOR REPLICATION ]  ( expresión_lógica )
 }
```

- ***nombre***: es el nombre de la restricción que se va a crear.
- ***otra_tabla***: es el nombre de la tabla a la que se hace referencia.
- ***campo_externo1***: son los nombres de los campos de la ***otra_tabla*** a los que se hace referencia.
- ***factor_relleno***: especifica cuánto se debe llenar cada página de índice utilizada para almacenar los datos de índice.Entre 0 y 100. Por defecto 0.
- ***grupo_ficheros***: indica dónde se almacena la tabla
- ***expresión_lógica***: Expresión que devuelve true o false

Restricción a nivel de tabla

```
CONSTRAINT nombre
 {      [ { PRIMARY KEY | UNIQUE }
          [ CLUSTERED | NONCLUSTERED ]
          { (principal1 [ ASC | DESC ] [ , principal2 [, ...] ] ) }
          [ WITH FILLFACTOR = factor_relleno]
          [ ON { grupo_ficheros | DEFAULT } ]
        ]     |
        FOREIGN KEY
          [ (referencia1 [, referencia2 [, ...] ] ) ]
          REFERENCES otra_tabla [ (campo_externo1 [ ,... campo_externo2 ] ) ]
          [ ON DELETE {NO ACTION | CASCADE | SET NULL | SET DEFAULT } ] [
          ON UPDATE {NO ACTION | CASCADE | SET NULL | SET DEFAULT } ] [
          NOT FOR REPLICATION ]    |
        CHECK [ NOT FOR REPLICATION ]  (expresión_lógica )
 }
```

- ***nombre***: es el nombre de la restricción que se va a crear.
- ***principal1, principal2***: son los nombres de los campos que compondrán la clave principal.
- ***referencia1, referencia2***: son los nombres de los campos que hacen referencia a otros de otra tabla.
- ***otra_tabla***: es el nombre de la tabla a la que se hace referencia.
- ***campo_externo1, campo_externo2***: son los nombres de los campos de la ***otra_tabla*** a los que se hace referencia.
- ***expresión_lógica***: criterio que se ha de cumplir. Devuelve true o false

CREAR TABLAS

Una base de datos almacena su información en tablas.

Para ver las tablas existentes creadas por los usuarios en una base de datos usamos el procedimiento almacenado "sp_tables @table_owner='dbo';":

```
USE JARDINERIA
GO
sp_tables @table_owner='dbo';
```

El parámetro @table_owner='dbo' indica que solo muestre las tablas de usuarios y no las que crea el SQL Server para administración interna.

Al crear una tabla debemos resolver qué campos (columnas) tendrá y que tipo de datos almacenarán cada uno de ellos, es decir, su estructura.

Las tablas son las unidades principales de almacenamiento en la bases de datos relacionales. La instrucción que se utiliza para crear una nueva tabla es:

CREATE TABLE nombre_tabla
(Descripción_Columna1 [, ... , Descripción_ColumnaN)) [,
Grupo_Restricciones_de_Tabla])

A continuación se explica cada una de las partes de esta sentencia:

Descripción_Columna

En este apartado se describe el nombre de la columna, junto con el tipo de datos que va a almacenar. De modo opcional, si es necesario, se incluyen las restricciones de columna que deseemos.

Grupo_Restricciones_de_Tabla

Indica las restricciones de tablas que se desean añadir, como puede ser la clave principal, foránea o externa, etc...

Tipos de datos.
Para cada columna añadida en la tabla se debe indicar el tipo de dato que va almacenará.

Creamos una tabla llamada "usuarios" y entre paréntesis definimos los campos y sus tipos:

```
create table usuarios (
 nombre varchar(30),
 clave varchar(10)
);
```

Cada campo con su tipo debe separarse con comas de los siguientes, excepto el último.

Cuando se crea una tabla debemos indicar su nombre y definir al menos un campo con su tipo de dato. En esta tabla "usuarios" definimos 2 campos:

- nombre: que contendrá una cadena de caracteres de 30 caracteres de longitud, que almacenará el nombre de usuario y
- clave: otra cadena de caracteres de 10 de longitud, que guardará la clave de cada usuario.

Para nombres de tablas, se puede utilizar cualquier carácter permitido para nombres de directorios, el primero debe ser un carácter alfabético y no puede contener espacios. La longitud máxima es de 128 caracteres.

Si intentamos crear una tabla con un nombre ya existente (existe otra tabla con ese nombre), mostrará un mensaje indicando que ya hay un objeto llamado 'usuarios' en la base de datos y la sentencia no se ejecutará.

Para ver la estructura de una tabla usamos el procedimiento almacenado "sp_columns" junto al nombre de la tabla:

```
sp_columns usuarios;
```

Aparece mucha información que no analizaremos en detalle, como el nombre de la tabla, su propietario, los campos, el tipo de dato de cada campo, su longitud, etc.:

```
...COLUMN_NAME     TYPE_NAME    LENGHT
_____________________________________
   nombre    varchar            30
   clave     varchar            10
```

Restricciones.

Las restricciones (constraints) son un método para mantener la integridad de los datos, asegurando que los valores ingresados sean válidos y que las relaciones entre las tablas se mantengan. Se pueden establecer a nivel de campos y de tablas.

Pueden definirse al crear la tabla ("create table") o agregarse a una tabla existente (empleando "alter table") y se pueden aplicar a un campo o a varios. Se aconseja crear las tablas y luego agregar las restricciones.

Se pueden crear, modificar y eliminar las restricciones sin eliminar la tabla y volver a crearla.

El procedimiento almacenado del sistema "sp_helpconstraint" junto al nombre de la tabla, nos muestra información acerca de las restricciones de dicha tabla.

```
sp_helpconstraint libros;
```

Aparecen varias columnas con la siguiente información:

- constraint_type: el tipo de restricción y sobre qué campo está establecida (DEFAULT on column autor),
- constraint_name: el nombre de la restricción (DF_libros_autor),
- delete_action y update_action: no tienen valores para este tipo de restricción.
- status_enabled y status_for_replication: no tienen valores para este tipo de restricción.
- constraint_keys: el valor por defecto (Desconocido).

Cuando se agrega una restricción a una tabla, SQL Server comprueba los datos existentes.

Como hemos visto tenemos grupos de restricciones que podemos separar en dos tipos, en función de si esas restricciones se aplican a columnas o tablas. Dentro de las restricciones podemos tener:

Restricciones a nivel de columna

Como se ha visto en la sintaxis, en la descripción de cada columna se indica el nombre de la columna, el tipo de dato que almacenará y opcionalmente un grupo de restricciones. En la siguiente tabla se puede ver las restricciones que tenemos a nuestra disposición para columnas:

Restricción	**Descripción**
NOT NULL	La columna no permite almacenar valores nulos.
UNIQUE	La columna almacenará valores exclusivos, es decir no podrá tener valores repetidos.
PRIMARY KEY	La columna compone la clave primaria, y por lo tanto no podrá almacenar ni valores nulos, ni valores repetidos. Puede tener una restricción "check" además. Cuando establecemos una clave primaria al definir la tabla, automáticamente SQL Server redefine el campo como "not null"; pero al agregar una restricción "primary key", los campos que son clave primaria DEBEN haber sido definidos "not null" (o ser implícitamente "not null" si se definen identity).

REFERENCES	La columna compone la clave externa de la correspondiente columna de la tabla que se indique.
CHECK(condiciones)	Los valores de esa columna deberán cumplir las condiciones que se le indiquen. Verifica los datos cada vez que se ejecuta una sentencia "insert" o "update". La condición puede hacer referencia a otros campos de la misma tabla. Un campo puede tener varias restricciones "check" y una restricción "check" puede incluir varios campos. Las condiciones para restricciones "check" también pueden incluir un patrón o una lista de valores.
DEFAULT(valor/función/NULL)	La cláusula DEFAULT especifica qué valores almacenará una columna en caso de que no se le indique ninguno. Es muy utilizado junto con la restricción NOT NULL para evitar almacenar valores erróneos. Se puede establecer uno por campo y no se puede emplear junto con la propiedad "identity". Acepta valores tomados de funciones del sistema, por ejemplo, podemos establecer que el valor por defecto de un campo de tipo datetime sea "getdate()"

EJEMPLOS

DEFAULT

```
create table libros(
  ...
  autor varchar(30) default 'Desconocido',
  ...
 );
```

CHECK

```
...
 check (CAMPO like '[A-Z][A-Z][0-9][0-9]');
```

O establecer que cierto campo asuma sólo los valores que se listan:

```
 ...
 check (CAMPO in ('lunes','miercoles','viernes'));
```

NOT NULL

```
CREATE TABLE PersonsNotNull(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

UNIQUE

```
CREATE TABLE Persons(
P_Id int NOT NULL UNIQUE,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

PRIMARY KEY

```
CREATE TABLE Persons(
P_Id int NOT NULL PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255)
)
```

FOREING KEY

```
CREATE TABLE Orders(
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int FOREIGN KEY REFERENCES Persons(P_Id)
)
```

La columna "P_Id" en la tabla "Persons" es la PRIMARY KEY en la tabla "Persons".

La columna "P_Id" en la tabla "Orders" es la FOREIGN KEY en la tabla "Orders".

Restricciones a nivel de tabla.

Del mismo modo que indicamos restricciones a nivel de columna es posible indicar las restricciones a nivel de tabla. Las restricciones que se pueden indicar para una tabla son las siguientes:

Restricción	Descripción
UNIQUE(Columna1, ..., ColumnaN)	Las columnas indicadas no pueden almacenar valores duplicados. Es común utilizar esta restricción a nivel de tabla para añadir claves alternativas
PRIMARY KEY (Columna1, ..., ColumnaN)	Las columnas indicadas no pueden almacenar ni valores nulos, ni duplicados, ya que serán clave principal o primaria.
FOREIGN KEY (Columna1, ..., ColumnaN) REFERENCES Tabla (ColumnaB1, ..., ColumnaBN)	Las columnas definidas son la clave externa correspondiente a la clave primaria que forman el conjunto de columnas "B" de la tabla definida
CHECK(condiciones)	La tabla deberá cumplir las condiciones que se le indiquen

NOTA. Las restricciones CHECK, UNIQUE, PRIMARY KEY y FOREIGN KEY pueden ser aplicadas para columnas en caso de que sólo afecte a una única columna. Si estas restricciones afectan a varias columnas obligatoriamente deberemos definirlas a nivel de tabla.

EJEMPLOS

UNIQUE

```
CREATE TABLE Persons(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT uc_PersonID UNIQUE (P_Id,LastName)
)
```

PRIMARY KEY

```
CREATE TABLE Persons(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
)
```

La PRIMARY KEY es pk_PersonID y está formada por dos campos: P_Id y LastName

FOREING KEY

```
CREATE TABLE Orders(
O_Id int NOT NULL PRIMARY KEY,
OrderNo int NOT NULL,
P_Id int,
CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)REFERENCES Persons(P_Id)

)
```

CHECK

```
CREATE TABLE Persons
(
P_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Address varchar(255),
City varchar(255),
CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')
)
```

EJEMPLO RESUMEN

En el siguiente ejemplo creamos la tabla "libros" con varias restricciones:

```
 create table libros(
  codigo int identity,
  titulo varchar(40),
  codigoautor int not null,
  codigoeditorial tinyint not null,
  precio decimal(5,2) constraint DF_precio default (0),
  constraint PK_libros_codigo  primary key clustered (codigo),
  constraint UQ_libros_tituloautor unique (titulo,codigoautor),
```

```
  constraint FK_libros_editorial foreign key (codigoeditorial)references editoriales(codigo)
   on update cascade,
  constraint FK_libros_autores foreign key (codigoautor)references autores(codigo)
   on update cascade,
  constraint CK_precio_positivo check (precio>=0)
);

En el ejemplo anterior creamos:

   • Una restricción "default" para el campo "precio" (restricción a nivel de campo);
   • Una restricción "primary key" con índice agrupado para el campo "codigo" (a nivel de tabla);
   • Una restricción "unique" con índice no agrupado (por defecto) para los campos "titulo" y "codigoautor" (a
     nivel de tabla);
   • Una restricción "foreign key" para establecer el campo "codigoeditorial" como clave externa que haga
     referencia al campo "codigo" de "editoriales y permita actualizaciones en cascada y no eliminaciones (por
     defecto "no action");
   • Una restricción "foreign key" para establecer el campo "codigoautor" como clave externa que haga
     referencia al campo "codigo" de "autores" y permita actualizaciones en cascada y no eliminaciones;
   • Una restricción "check" para el campo "precio" que no admita valores negativos;
```

Si definimos una restricción "foreign key" al crear una tabla, la tabla referenciada debe existir. Por eso se aconseja crear las tablas primero y luego establecer restricciones.

CAMPOS CALCULADOS

Un campo calculado es un campo que no se almacena físicamente en la tabla. SQL Server emplea una fórmula que detalla el usuario al definir dicho campo para calcular el valor según otros campos de la misma tabla.

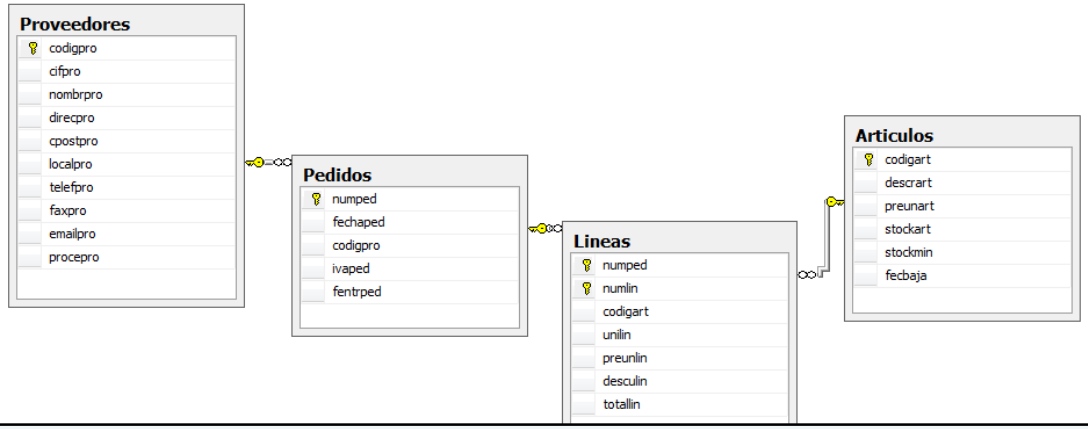
Un campo calculado no puede:

- Definirse como "not null".
- Ser una subconsulta.
- Tener restricción "default" o "foreign key".
- Insertarse ni actualizarse.

Puede ser empleado como parte de restricciones "primary key" o "unique" si la expresión que la define no cambia en cada consulta. Los campos de los cuales depende el campo calculado no pueden eliminarse, se debe eliminar primero el campo calculado.

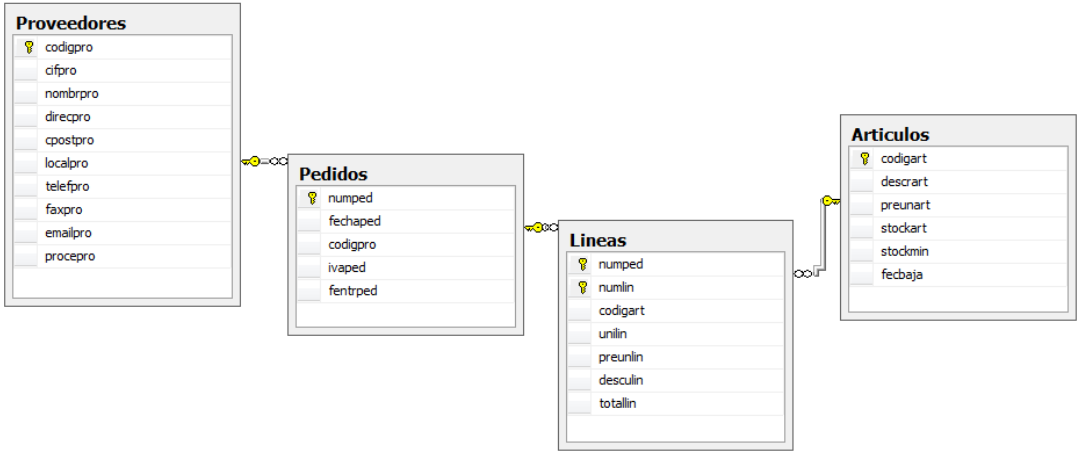
Ejemplo: Creamos un campo calculado denominado "sueldototal" que suma al sueldo básico de cada empleado la cantidad abonada por los hijos (100 por cada hijo):

```
create table empleados(
 documento char(8),
 nombre varchar(10),
 domicilio varchar(30),
 sueldobasico decimal(6,2),
 cantidadhijos tinyint default 0,
 sueldototal as sueldobasico + (cantidadhijos*100)
);
```



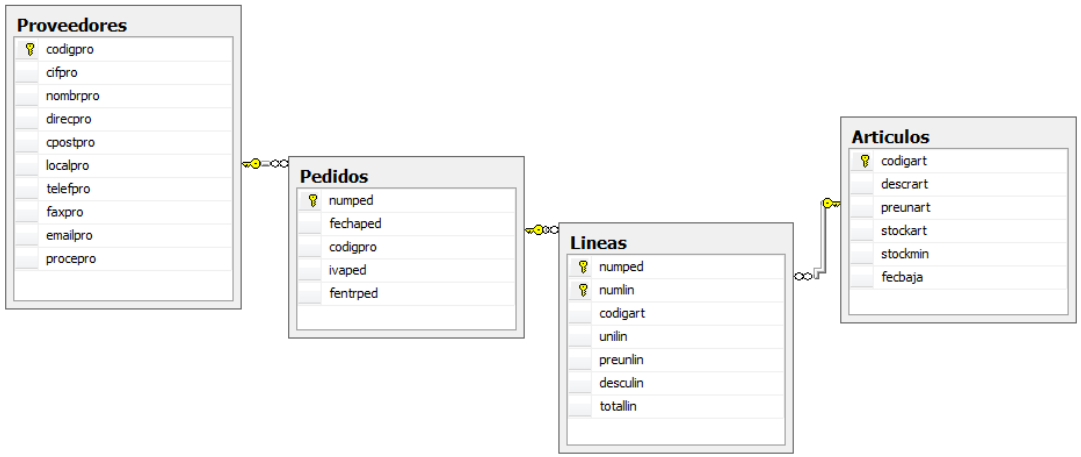

```
CREATE TABLE Proveedores (
 codigpro        CHAR(4)   NOT NULL   CONSTRAINT id_pro PRIMARY KEY,
 cifpro          CHAR(12)  NOT NULL   CONSTRAINT u_cif UNIQUE,
 nombrpro        CHAR(30)  NOT NULL,
 direcpro        CHAR(30)  NOT NULL,
 cpostpro        CHAR(5)    NOT NULL CHECK (cpostpro like '[0-9][0-9][0-9][0-9][0-9]'),
 localpro         CHAR(20)  NOT NULL,
 telefpro         CHAR(17)  NOT NULL,
 faxpro          CHAR(17),
 emailpro        CHAR(25),
 procepro        CHAR(5)  NOT NULL CHECK (procepro in ('UE', 'No UE')))
```

CREATE TABLE: EJEMPLOS (y 2)



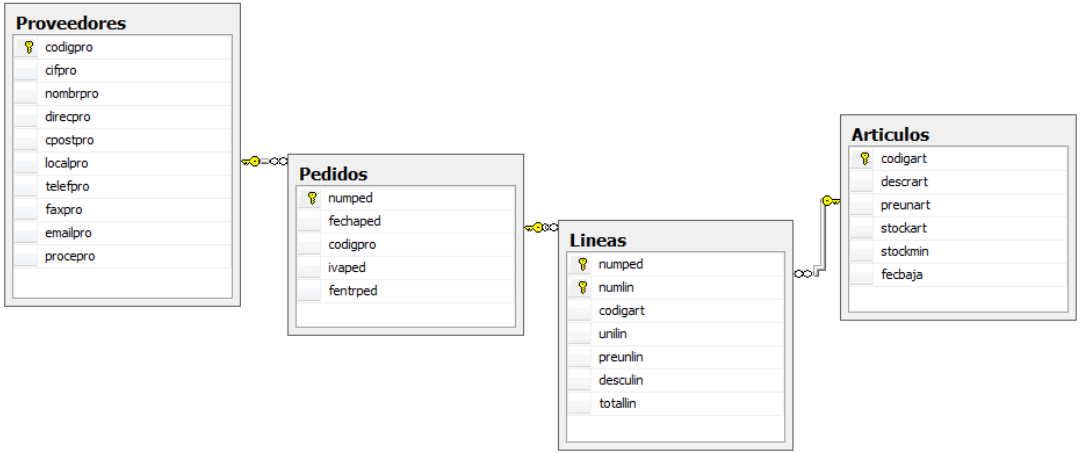

```
CREATE TABLE Articulos (
   codigart          CHAR(6)   NOT NULL   CONSTRAINT id_art PRIMARY KEY,
   descrart          CHAR(40) NOT NULL,
   preunart          DECIMAL(9,2)  NOT NULL,
   stockart           INTEGER  NOT NULL CHECK (stockart >=0),
   stockmin           INTEGER  NOT NULL CHECK (stockmin>=0),
   fecbaja     DATE)
```

CREATE TABLE: EJEMPLOS (y 3)




```
CREATE TABLE Pedidos (
   numped        INTEGER        NOT NULL   CONSTRAINT id_ped PRIMARY KEY,
   fechaped      DATE           NOT NULL   DEFAULT getdate(),
   codigpro      CHAR(4)        NOT NULL,
   ivaped        DECIMAL(4,1)   NOT NULL CHECK (ivaped>0 and ivaped<100),
   fentrped      DATE           NOT NULL,
         CONSTRAINT f_pro FOREIGN KEY (codigpro) REFERENCES Proveedores (codigpro),
         CONSTRAINT c_fecha CHECK (fechaped<=fentrped))
```

CREATE TABLE: EJEMPLOS (y 4)




```
CREATE TABLE Lineas (
   numped          INTEGER NOT NULL,
   numlin          SMALLINT NOT NULL,
   codigart        CHAR(6)  NOT NULL,
   unilin          DECIMAL (6,0)      NOT NULL,
   preunlin        DECIMAL (9,2)      NOT NULL,
   desculin        DECIMAL (4,1)      NOT NULL CHECK (desculin>=0 and desculin<=100),
   totallin     AS ([preunlin] * [unilin] * (1 –[desculin] / 100)),
        CONSTRAINT id_lin PRIMARY KEY (numped, numlin),
        CONSTRAINT f_ped FOREIGN KEY (numped) REFERENCES Pedidos (numped),
        CONSTRAINT f_art FOREIGN KEY (codigart) REFERENCES Articulos (codigart))
```

ALTER TABLE

Para modificar el diseño de una tabla que ya existe en la base de datos. Podemos utilizarla para agregar, modificar y eliminar campos de una tabla. Su sintaxis es:

```
ALTER TABLE tabla
 {
  ALTER COLUMN  { campo   tipo [(tamaño)] }  [COLLATE] [ NULL | NOT NULL]  |
  ADD
     {<column_definition> | <computed_definition>|<table_constraint>} [,…n]          |
 DROP
     {COLUMN nombre_campo   |  CONSTRAINT nombre_restricción} [,…n]          |
  { ENABLE | DISABLE } TRIGGER  { ALL | nombre_trigger [ ,...n ] }                    |
  { CHECK | NOCHECK } CONSTRAINT { ALL | nombre_restricción[ ,...n ] }}
```

I. Para agregar un nuevo campo a una tabla empleamos la siguiente sintaxis básica:

```
alter table NOMBRETABLA
 add NOMBRENUEVOCAMPO DEFINICION;
```

En el siguiente ejemplo agregamos el campo "cantidad" a la tabla "libros", de tipo tinyint, que acepta valores nulos:

```
alter table libros
 add cantidad tinyint;
```

SQL Server no permite agregar campos "not null" a menos que se especifique un valor por defecto:

```
alter table libros
 add autor varchar(20) not null default 'Desconocido';
```

Al agregar un campo puede especificarse que sea "identity" (siempre que no exista otro campo identity).

Para eliminar campos de una tabla la sintaxis básica es la siguiente:

```
alter table NOMBRETABLA
drop column NOMBRECAMPO;
```

En el siguiente ejemplo eliminamos el campo "precio" de la tabla "libros":

```
alter table libros
drop column precio;
```

No pueden eliminarse los campos que son usados por un índice o que tengan restricciones. No puede eliminarse un campo si es el único en la tabla.

Podemos eliminar varios campos en una sola sentencia:

```
alter table libros
drop column editorial,edicion;
```

II. Para modificar campos de una tabla.

La sintaxis básica para modificar un campo existente es la siguiente:

```
alter table NOMBRETABLA
alter column CAMPO NUEVADEFINICION;
```

Modificamos el campo "titulo" extendiendo su longitud y para que NO admita valores nulos:

```
alter table libros
alter column titulo varchar(40) not null;
```

En el siguiente ejemplo alteramos el campo "precio" de la tabla "libros" que fue definido "decimal(6,2) not null" para que no acepte valores nulos:

```
alter table libros
alter column precio decimal(6,2) null;
```

SQL Server tiene algunas excepciones al momento de modificar los campos. No permite modificar:

- Campos de tipo text, image, ntext y timestamp.
- Un campo que es usado en un campo calculado.
- Campos que son parte de índices o tienen restricciones, a menos que el cambio no afecte al índice o a la restricción, por ejemplo, se puede ampliar la longitud de un campo de tipo carácter.
- Campos que afecten a los datos existentes cuando una tabla contiene registros (ejemplo: un campo contiene valores nulos y se pretende redefinirlo como "not null"; un campo int guarda un valor 300 y se pretende modificarlo a tinyint, etc.).

III. Para agregar restricciones.

DEFAULT

Podemos agregar una restricción "default" a una tabla existente con la sintaxis básica siguiente:

```
alter table NOMBRETABLA
add constraint NOMBRECONSTRAINT  default VALORPORDEFECTO
for CAMPO;
```

En la siguiente sentencia agregamos una restricción "default" al campo autor de la tabla existente "libros", que almacena el valor "Desconocido" en dicho campo si no ingresamos un valor en un "insert":

```
alter table libros
add constraint DF_libros_autor default 'Desconocido'
for autor;
```

Por convención, cuando demos el nombre a las restricciones "default" emplearemos un formato similar al que le da SQL Server: "DF_NOMBRETABLA_NOMBRECAMPO".

CHECK

La sintaxis básica es la siguiente:

```
alter table NOMBRETABLA
add constraint NOMBRECONSTRAINT
check CONDICION;
```

Trabajamos con la tabla "libros" de una librería que tiene los siguientes campos: codigo, titulo, autor, editorial, preciomin (que indica el precio para los minoristas) y preciomay (que indica el precio para los mayoristas).

Los campos correspondientes a los precios (minorista y mayorista) se definen de tipo decimal(5,2), es decir, aceptan valores entre -999.99 y 999.99. Podemos controlar que no se ingresen valores negativos para dichos campos agregando una restricción "check":

```
alter table libros
add constraint CK_libros_precio_positivo
check (preciomin>=0 and preciomay>=0);
```

Si la tabla contiene registros que no cumplen con la restricción que se va a establecer, la restricción no se puede establecer, hasta que todos los registros cumplan con dicha restricción.

La condición puede hacer referencia a otros campos de la misma tabla. Por ejemplo, podemos controlar que el precio mayorista no sea mayor al precio minorista:

```
alter table libros
add constraint CK_libros_preciominmay  check (preciomay<=preciomin);
```

Por convención, cuando demos el nombre a las restricciones "check" seguiremos la misma estructura: comenzamos con "CK", seguido del nombre de la tabla, del campo y alguna palabra con la cual podamos identificar fácilmente de qué se trata la restricción, por si tenemos varias restricciones "check" para el mismo campo.

PRIMARY KEY

Podemos agregar una restricción "primary key" a una tabla existente con la sintaxis básica siguiente:

```
alter table NOMBRETABLA
add constraint NOMBRECONSTRAINT primary key (CAMPO,...);
```

En el siguiente ejemplo definimos una restricción "primary key" para nuestra tabla "libros" para asegurarnos que cada libro tendrá un código diferente y único:

```
alter table libros
add constraint PK_libros_codigo primary key(codigo);
```

Por convención, cuando demos el nombre a las restricciones "primary key" seguiremos el formato "PK_NOMBRETABLA_NOMBRECAMPO".

UNIQUE

La sintaxis general es la siguiente:

```
alter table NOMBRETABLA
add constraint NOMBRERESTRICCION unique (CAMPO);
```

Ejemplo:

```
alter table alumnos
  add constraint UQ_alumnos_documento  unique (documento);
```

En el ejemplo anterior se agrega una restricción "unique" sobre el campo "documento" de la tabla "alumnos", esto asegura que no se pueda ingresar un documento si ya existe. Esta restricción permite valores nulos, así que si se ingresa el valor "null" para el campo "documento", se acepta.

Por convención, cuando demos el nombre a las restricciones "unique" seguiremos la misma estructura: "UQ_NOMBRETABLA_NOMBRECAMPO".

Cuando agregamos una restricción a una tabla que contiene información, SQL Server controla los datos existentes para confirmar que cumplen la condición de la restricción, si no los cumple, la restricción no se aplica y aparece un mensaje de error. En el caso del ejemplo anterior, si la tabla contiene números de documento duplicados, la restricción no podrá establecerse; si podrá establecerse si tiene valores nulos.

CAMPO CALCULADO

También se puede agregar un campo calculado a una tabla existente:

```
alter table NOMBRETABLA
 add NOMBRECAMPOCALCULADO as EXPRESION;

alter table empleados
 add sueldototal as sueldo+(cantidadhijos*100);
```

FOREING KEY

La sintaxis básica para agregar la restriccción "foreign key" es la siguiente:

```
alter table NOMBRETABLA1
 with OPCIONDECHEQUEO
 add constraint NOMBRECONSTRAINT
 foreign key (CAMPOCLAVEFORANEA)
 references NOMBRETABLA2 (CAMPOCLAVEPRIMARIA)
 on update OPCION
 on delete OPCION;
```

NOTA:

Sabemos que si agregamos una restricción a una tabla que contiene datos, SQL Server los controla para asegurarse que cumplen con la restricción. Si no la cumplen no se puede añadir la restricción. Pero es posible deshabilitar esta comprobación en las restricciones "check" y "foreign key".

Podemos hacerlo al momento de agregar la restricción a una tabla con datos, incluyendo la opción "with nocheck" en la instrucción "alter table"; si se emplea esta opción, los datos no van a cumplir la restricción.

La opción "with OPCIONDECHEQUEO" especifica si se controlan los datos existentes o no con "check" y "nocheck" respectivamente. Por defecto, si no se especifica, la opción es "check".

En el siguiente ejemplo se agrega una restricción "foreign key" que controla que todos los códigos de editorial tengan un código válido, es decir, dicho código exista en "editoriales". La restricción no se aplica en los datos existentes pero si en los siguientes ingresos, modificaciones y actualizaciones:

```
alter table libros
with nocheck
add constraint FK_libros_codigoeditorial
foreing key (codigoeditorial)
references editoriales(codigo);
```

ALTER TABLE: ejemplos

```
ALTER TABLE dbo.Pedidos WITH NOCHECK ADD
   CONSTRAINT DF_Pedidos_fechaped DEFAULT (getdate()) FOR fechaped,
   CONSTRAINT c_fecha CHECK (fechaped <= fentrped),
   CHECK (ivaped > 0 and ivaped < 100)
GO


ALTER TABLE dbo.Proveedores ADD
    CHECK (cpostpro like '[0-9][0-9][0-9][0-9][0-9]'),
    CHECK (procepro = 'No UE' or procepro = 'UE')
GO
```

ELIMINAR UNA TABLA DROP TABLE

Para eliminar una tabla de una base de datos tenemos la sentencia DROP TABLE. Con ella quitamos una o varias definiciones de tabla y todos los datos, índices, desencadenadores, restricciones y especificaciones de permisos que tengan esas tablas.

Las vistas o procedimientos almacenados que hagan referencia a la tabla quitada se deben quitar explícitamente con DROP VIEW o DROP PROCEDURE.
Su sintaxis es:

DROP TABLE [nbBaseDatos.[nbEsquema].|nbEsquema.]nbTabla[,...n] [;]

Para que las reglas de integridad referencial se cumplan, no se puede eliminar una tabla señalada por una restricción FOREIGN KEY. Primero se debe quitar la restricción FOREIGN KEY o la tabla que tiene la clave ajena.

Se pueden quitar varias tablas de cualquier base de datos en una misma sentencia DROP TABLE. Se irán eliminando en el mismo orden en que aparecen en la lista por lo que podremos eliminar dos tablas relacionadas con una sola sentencia pero escribiendo la tabla que contiene la clave ajena primero y después la tabla principal.

Ejemplo:

DROP TABLE mitabla; -- Elimina la tabla miTabla tanto su definición como los datos, índices definidos sobre ella y permisos.

Si intentamos eliminar una tabla que no existe, aparece un mensaje de error indicando tal situación y la sentencia no se ejecuta. Para evitar este mensaje podemos agregar a la instrucción lo siguiente:

```
if object_id('usuarios') is not null
drop table usuarios;
```

En la sentencia precedente especificamos que elimine la tabla "usuarios" si existe.

VISTAS

Las vistas son consultas almacenadas (tablas lógicas). Aunque también se pueden definir físicas o materializadas. Para crear una vista :

CREATE VIEW [< *nombreBD* > .] [< *propietario* > .] *nombre* [(*campo* [,...n])]
[WITH <*view_attribute*> [,...n]]
AS
Instrucción_Select
[WITH CHECK OPTION]

- ***nombreBD**:* es el nombre de la base de datos en la que se crea.
- ***propietario***: cuenta de usuario que crea la vista
- ***nombre**:* es el nombre de la vista que se va a crear.
- ***campo**: e*s el nombre que se va a utilizar para una columna en una vista.
- ***instrucción_Select:*** consulta a través de la cuál se define la vista
- ***view_attribute:*** toma uno de los siguientes valores

[ENCRYPTION]: evita que la vista se publique como parte de la réplica de SQL Server

[SCHEMABINDING]: enlaza la vista al esquema de las tablas subyacentes. Cuando se especifica, las tablas base no se pueden modificar de una forma que afecte a la definición de la vista.

[VIEW_METADATA]: Especifica que la instancia de SQL Server devolverá a las API de DB-Library, ODBC y OLE DB la información de metadatos sobre la vista en vez de las tablas base.

[WITH CHECK OPTION] : Exige que todas las instrucciones de modificación de datos ejecutadas contra la vista se adhieran a los criterios establecidos en la instrucción

Una vista es una alternativa para mostrar datos de varias tablas. Una vista es como una tabla virtual que almacena una consulta. Los datos accesibles a través de la vista no están almacenados en la base de datos como un objeto.

Por lo tanto, una vista almacena una consulta como un objeto para utilizarse posteriormente. Las tablas consultadas en una vista se llaman tablas base. En general, se puede dar un nombre a cualquier consulta y almacenarla como una vista.

Una vista suele llamarse también tabla virtual porque los resultados que retorna y la manera de referenciarlas es la misma que para una tabla.

Las vistas permiten:

- Ocultar información: permitiendo el acceso a algunos datos y manteniendo oculto el resto de la información que no se incluye en la vista. El usuario opera con los datos de una vista como si se tratara de una tabla, pudiendo modificar tales datos.
- Simplificar la administración de los permisos de usuario: se pueden dar al usuario permisos para que solamente pueda acceder a los datos a través de vistas, en lugar de concederle permisos para acceder a ciertos campos, así se protegen las tablas base de cambios en su estructura.
- Mejorar el rendimiento: se puede evitar teclear instrucciones repetidamente almacenando en una vista el resultado de una consulta compleja que incluya información de varias tablas.

Igual que sucede con una tabla, se pueden insertar, actualizar, borrar y seleccionar datos en una vista. Siempre se van a poder seleccionar datos en una vista, aunque existen ciertas restricciones para realizar el resto de las operaciones sobre vistas.

Podemos crear vistas con:

- un subconjunto de registros y campos de una tabla;
- una unión de varias tablas;
- una combinación de varias tablas;
- un subconjunto de otra vista, combinación de vistas y tablas.

Una vista se define usando un "select".

La sintaxis básica parcial para crear una vista es la siguiente:

```
create view NOMBREVISTA as
 SENTENCIASSELECT
  from TABLA;
```

El contenido de una vista se muestra con un "select":

```
select *from NOMBREVISTA;
```

En el siguiente ejemplo creamos la vista "vista_empleados", que es resultado de una combinación en la cual se muestran 4 campos:

```
create view vista_empleados as
 select (apellido+' '+e.nombre) as nombre,sexo,
  s.nombre as seccion, cantidadhijos
  from empleados as e
  join secciones as s
  on codigo=seccion
```

Para ver la información contenida en la vista creada anteriormente escribimos:

```
select *from vista_empleados;
```

Podemos realizar consultas a una vista como si se tratara de una tabla:

```
select seccion,count(*) as cantidad
 from vista_empleados;
```

Los campos y expresiones de la consulta que define una vista DEBEN tener un nombre. Se debe colocar nombre de campo cuando es un campo calculado o si hay 2 campos con el mismo nombre. En el ejemplo, al concatenar los campos "apellido" y "nombre" colocamos un alias; si no lo hubiésemos hecho aparecería un mensaje de error porque dicha expresión DEBE tener un encabezado, SQL Server no lo coloca por defecto.

Los nombres de los campos y expresiones de la consulta que define una vista DEBEN ser únicos (no puede haber dos campos o encabezados con igual nombre). En la vista

definida en el ejemplo, al campo "s.nombre" le colocamos un alias porque ya había un encabezado (el alias de la concatenación) llamado "nombre" y no pueden repetirse, si sucediera, aparecería un mensaje de error.

Se dispone también de los comandos DROP VIEW (eliminar una vista) y ALTER VIEW (para hacer modificaciones en la definición de la vista)

Un ejemplo de utilización de vistas podría ser: en la base de datos de un banco podríamos definir una vista con los datos de las cuentas de un determinado cliente. De este modo podríamos permitir a cada cliente acceder únicamente a sus datos.

VISTAS: ejemplo vista lógica

```
CREATE VIEW dbo.EncabezadoPedido
AS
SELECT     dbo.Pedidos.numped, dbo.Pedidos.fechaped, dbo.Pedidos.codigpro,
dbo.Pedidos.ivaped, dbo.Pedidos.fentrped, dbo.Proveedores.nombrpro,
dbo.Proveedores.direcpro, dbo.Proveedores.cpostpro, dbo.Proveedores.localpro,
dbo.Proveedores.telefpro, dbo.Proveedores.faxpro,
dbo.Proveedores.procepro, dbo.Proveedores.emailpro, dbo.Proveedores.cifpro
FROM       dbo.Proveedores INNER JOIN
                dbo.Pedidos ON dbo.Proveedores.codigpro = dbo.Pedidos.codigpro
```

VISTAS ACTUALIZABLES

Algunas vistas son actualizables. Esto significa que se pueden emplear en sentencias como UPDATE, DELETE o INSERT para actualizar el contenido de las tablas subyacentes. (Las actualizaciones se transfieren a la/s tabla/s original/es (con ciertas limitaciones)).

Una vista es actualizable si:

- El *SELECT* no tiene ninguna expresión de valor agregado ni especificación de DISTINCT
- Cualquier atributo que no aparezca en la cláusula SELECT puede definirse como nulo
- La consulta no tiene cláusulas GROUP BY ni HAVING
- Cualquier modificación, UPDATE, INSERT y DELETE, debe hacer referencia a las columnas de una única tabla base
- Las columnas que se van a modificar no se ven afectadas por las cláusulas GROUP BY, HAVING o DISTINCT.
- No usar JOIN externos. En el caso de inner JOIN podemos actualizar o insertar siempre y cuando los campos afectados sean únicamente los de una de las tablas implicadas en el join.

Por ejemplo, no podremos insertar registros en una vista que contenga columnas simples y derivadas, pero podemos actualizarla si actualizamos únicamente las columnas derivadas.

Ejemplo:

*CREATE VIEW v as SELECT a, 2*a as b FROM T1;*

Esta vista consiste en dos campos, uno de ellos derivado b.

Se podría modificar el campo no derivado:

UPDATE v SET a=a+1;

Pero no se podría modificar el campo b

UPDATE v SET b=b+1; Daría error

La cláusula WITH CHECK OPTION puede utilizarse para evitar inserciones o actualizaciones en registros distintos de los determinados en la cláusula WHERE incluida en la definición de la vista.
Ejemplo: Dada la siguiente relación de una base de datos:
Cocinero (nombre:varchar, edad: number, país:varchar)
Obtener una vista con, únicamente, los cocineros franceses

CREATE VIEW Cocineros_franceses as SELECT * FROM Cocinero
WHERE pais= Francia WITH CHECK OPTION;
El CHECK OPTION impide que yo pueda añadir cocineros que no sean franceses