

Creación de tablas y restricciones

Antes de meternos de lleno en los conceptos de tablas y restricciones, realizaremos un breve repaso sobre el modelo de base de datos relacional.

Relaciones

Como estudiamos anteriormente, cada entidad está representada lógicamente en una base de datos mediante una **tabla**. Y cada **registro** (o *tupla*) es una instancia de dicha entidad. Por ejemplo, si tenemos una tabla llamada **Ciudades** con cinco registros:

	IDCiudad	IDPaís	Ciudad
1	1	ARG	Buenos Aires
2	2	DEU	Berlín
3	3	ESP	Madrid
4	4	FRA	Paris
5	5	GBR	Londres

Significa que contamos con cinco instancias de ciudades distintas, que pueden ser o no capitales de un país, pueden pertenecer o no a un mismo país, etc. El aspecto fundamental en este caso (y en toda base de datos relacional) es poder asegurar que en una tabla **no se almacenarán registros duplicados**.

En la mayoría de los casos, con una única tabla no es posible representar el problema a resolver. Es por ello que en nuestras bases de datos tendremos por lo general una colección de tablas, y cada una de ellas representará una de las entidades de nuestro problema.

Si bien distintas tablas representan elementos distintos, es muy común que existan elementos en común entre algunas de ellas. Estos elementos en común relacionan las tablas entre sí. Es por eso que podemos decir que una **relación** es un vínculo existente entre dos o más entidades. Generalmente describe algún tipo de interacción entre las mismas.

Por ejemplo, una relación entre las entidades *Autores* y *Países* podría llamarse “nació en” ya que un autor tiene un país de origen.

Tablas

La información se almacena en una base de datos dentro de **tablas**. Las mismas se caracterizan por representarse gráficamente como objetos rectangulares conformados por **filas** y **columnas**. Cada columna almacenará información sobre una propiedad determinada de la tabla (**atributo**) y cada fila almacenará una instancia de la tabla en cuestión, es decir, un conjunto de información de cada uno de sus atributos:

atributo 1	atributo 2	atributo N
valor 1 fila 1	valor 2 fila 1	valor N fila 1
valor 1 fila 2	valor 2 fila 2	valor N fila 2
valor 1 fila 3	valor 2 fila 3	valor N fila 3
.	.	.
valor 1 fila M	valor 2 fila M	valor N fila M

Dominios

Los dominios determinan un conjunto de posibles valores válidos para un atributo. Cada dominio tiene un nombre y una posible definición del mismo. Por ejemplo:

Edad: número entero positivo entre 1 y 120 (1, 41, 38, 72, 49, 6...)

País de nacimiento: texto de 50 caracteres (Argentina, Chile, España, Italia, Brasil...)

Fecha de nacimiento: una fecha representada por día, mes y año (17/02/1984, 02/10/1986)

Nulos

Es común que en ocasiones necesitemos indicar que el contenido de un atributo equivale a “ningún valor”. El problema que surge aquí es que la ausencia de valor es difícil de representar entre los distintos tipos de datos. Es necesario poder identificar dicha ausencia de valor en atributos de tipo entero, real, cadena de texto, booleano, fecha, etc. Es por eso que surge el concepto de **nulo** (en inglés **NULL**). De esta manera podemos indicar que el contenido de un atributo no tiene ningún valor.

En claves foráneas, los valores nulos se utilizan para indicar que el registro actual no está relacionado con ningún otro registro. En otros atributos, indica que dicho valor podría no ser rellenado por alguna razón (por ejemplo, porque es desconocido).

Claves

Dentro de un modelo relacional existen distintos tipos de claves. Las claves nos dan la pauta acerca de si dicho valor representa unívocamente al registro dentro de la tabla o bien si representa un registro de manera unívoca pero en otra tabla.

Las claves más comunes y que utilizaremos más a menudo son:

- **Clave primaria (Primary Key o PK):** la clave primaria es aquella columna (o conjunto de columnas) que representa de manera unívoca a todo el registro. Es un identificador que será siempre único en toda la tabla. No acepta valores nulos.
- **Clave foránea (Foreign Key o FK):** la clave foránea es una columna (o conjunto de columnas) de una tabla, que está relacionada y es dependiente de una clave primaria (o alternativa) de otra tabla. Puede recibir valores nulos.

Otros tipos de claves son:

- ❖ **Clave candidata:** en una tabla, puede ocurrir que exista más de una columna que la representa unívocamente. De manera que se deberá elegir una de ellas para designarla como clave primaria, y el resto serán claves candidatas.
- ❖ **Clave alternativa:** la clave alternativa es cualquier clave candidata que no haya sido seleccionada como clave primaria.
- ❖ **Clave compuesta:** una clave compuesta es una clave que está conformada por más de una columna.

Restricciones

En términos de bases de datos, podemos definir a una **restricción** como una regla de validación que debe cumplirse para poder realizar cambios en los datos de una tabla (ya sea para insertar, modificar o eliminar registros). Los tipos de restricciones con los que trabajaremos en esta materia son los siguientes:

- **Clave primaria (Primary Key o PK):** como vimos anteriormente, los valores de una columna (o grupo de columnas) que está definida como clave primaria no pueden repetirse.

- **Clave única (Unique o UQ):** funciona igual que la clave primaria. La diferencia entre ambas es que sólo puede existir una clave primaria por tabla, mientras que claves únicas podemos definir tantas como sean necesarias.
- **Clave foránea (Foreign Key o FK):** los valores suministrados en estas columnas deben existir obligatoriamente en las columnas de las tablas a las que hacen referencia. A través de una clave foránea podemos relacionar una tabla con otra.
- **No nulidad (NOT NULL):** a las columnas definidas como NOT NULL se les debe suministrar obligatoriamente valores no nulos (es decir, no está permitida la ausencia de dato). Por ejemplo, en una tabla *Usuarios* donde cada registro se identifica unívocamente con un Id alfanumérico, y en la que se desea registrar además, y de forma obligatoria, los nombres y apellidos de los usuarios, debemos incorporar la condición NOT NULL a las columnas *Nombres* y *Apellidos* (suponiendo que así se llamaran las columnas que almacenan nombres y apellidos de usuarios respectivamente).
- **Check:** es una regla o condición (o conjunto de ellas) que debe cumplirse para incorporar o actualizar un valor en una columna. Por ejemplo, en la columna *Paginas* de una tabla *Libros*, sería recomendable añadir una restricción de tipo check para evitar el ingreso accidental de un número negativo (valor absurdo para lo que representa esa columna).
- **Default:** si bien no es una restricción en el sentido más estricto de la palabra, es una condición que podemos aplicar a una columna para que, ante la ausencia de dato, tome un valor predeterminado al insertar un registro. Es decir, a una columna que posee *default*, si no le proporcionamos un dato de forma explícita, se le asignará automáticamente el valor que establecimos por defecto y que, como es de esperar, debe ser del mismo tipo de dato que la columna.

Las ventajas de tener un valor por defecto en una columna, además de evitar la nulidad, es lograr consultas de inserción de registros más acotadas (al no tener que escribir repetitivamente un valor que, por lo general, es siempre el mismo) y cierta estandarización de valores (por ejemplo, en una tabla *Usuarios*, que posee una columna llamada *Estado* de tipo *varchar* que acepta dos posibles valores: "Activo" e "Inactivo", nos resultaría útil asignar en dicha columna el valor por defecto "Activo" al insertar un registro para generar un nuevo usuario. Esto ayuda a que, en futuras inserciones, no se represente el estado activo del usuario con distintos valores como por ejemplo "Act.", "A", etc.

Ejemplo de clave primaria

Supongamos que estamos insertando datos en una tabla llamada **Categorías** que representa un criterio de clasificación de libros. La misma consta de la columna *IDCategoría* (de tipo BIGINT) definida como clave primaria y de la columna *Categoría* de tipo VARCHAR:

CATEGORIAS	
IDCategoría (BIGINT) (PK)	Categoría (VARCHAR 50)
1	Novela
2	Ciencia ficción
3	Terror
1	Historia

Como podemos observar, el gestor de base de datos no permitirá el ingreso del cuarto registro (1, 'Historia') ya que repite el valor de otro registro en la columna que es clave primaria. A menos que proporcionemos un valor distinto de 1, 2 y 3 para la columna IDCategoría, no podremos insertar o modificar este registro.

Ejemplo de clave foránea

Para ejemplificar este tipo de restricción contamos con las tablas Países y Ciudades, que están definidas de la siguiente manera:

PAISES	
IDPais (VARCHAR 3) (PK)	Pais (VARCHAR 100)
ARG	Argentina
DEU	Alemania
ESP	España

CIUDADES		
IDCiudad (BIGINT) (PK)	IDPais (VARCHAR 3) (FK)	Ciudad (VARCHAR 100)
1	ARG	Buenos Aires
2	DEU	Berlín
3	EGY	El Cairo

La columna **IDPais** de la tabla Ciudades está definida como clave foránea. Es decir que dicha columna relaciona a la tabla Ciudades con la tabla Países. Por este motivo, cada registro que insertemos o modifiquemos en Ciudades debe poseer, en la columna IDPais, un valor que exista en la columna IDPais de Países. De lo contrario, como vemos en este caso con el valor 'EGY', el gestor de base de datos impedirá la operación: a menos que insertemos previamente un registro para el país Egipto con IDPais = 'EGY' en la tabla Países, no podremos crear un registro con IDPais = 'EGY' en la tabla Ciudades.

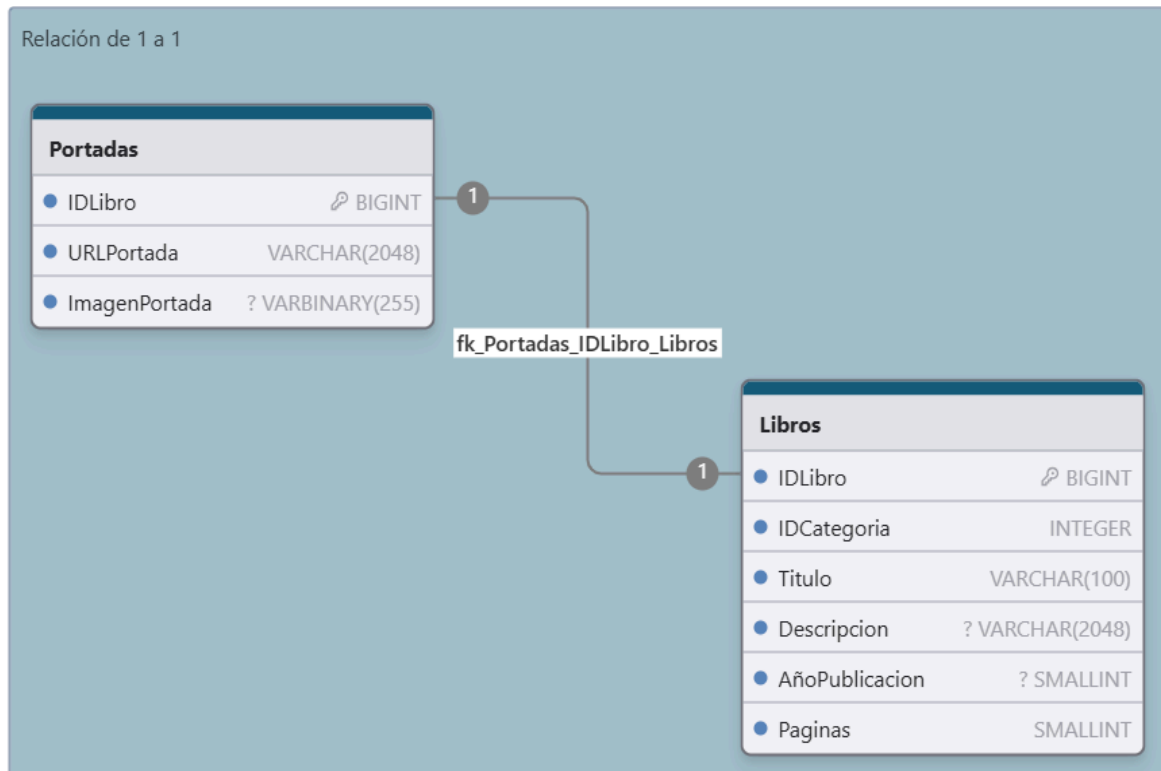
El concepto que encierra este ejemplo, y que garantiza que la base de datos sea íntegra y consistente, es conocido como **integridad referencial**, y es fundamental en las bases de datos relacionales. Se hace presente en toda operación de manipulación de datos (inserción, modificación y eliminación).

Hemos visto un ejemplo con alta y modificación. Respecto de la eliminación de un registro, la integridad referencial funciona de manera inversa a la inserción y modificación: para poder eliminar un registro que está relacionado con un registro de otra tabla, es necesario eliminar previamente todas las referencias al mismo, es decir, las filas que contengan una clave foránea que haga referencia a él. Siguiendo con este ejemplo, si quisiéramos eliminar el registro ('DEU', 'Alemania') de la tabla Países, tendremos que eliminar primero el registro (2, 'DEU', 'Berlín') en la tabla Ciudades. Caso contrario, el gestor de base de datos emitirá un error, ya que se estaría infringiendo la restricción de clave foránea (y por ende, la integridad referencial).

Tipos de relaciones

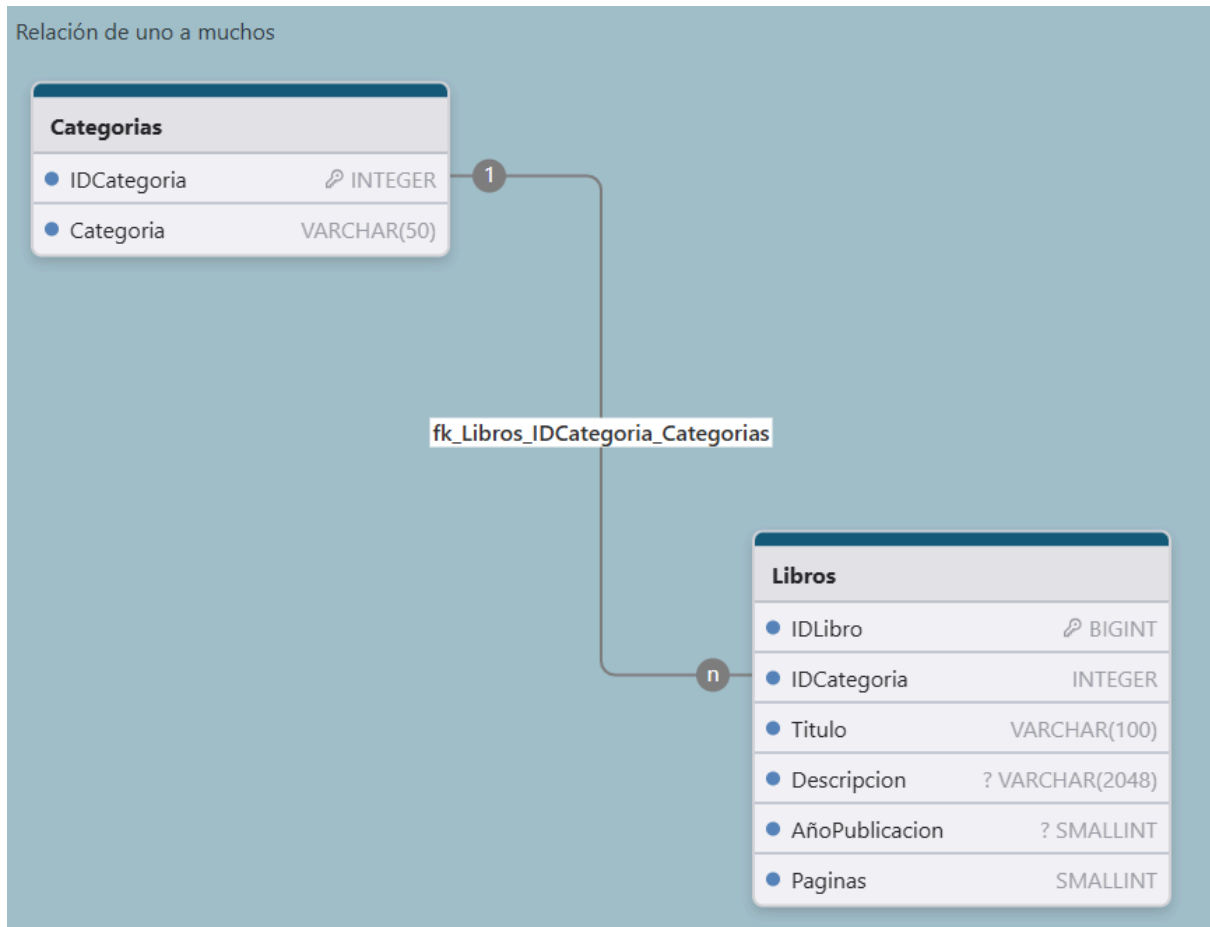
A continuación describiremos los tipos de relaciones que pueden existir entre diferentes tablas dentro de una base de datos relacional. Para los ejemplos utilizaremos una base de datos que almacenará un conjunto de información de una librería (tienda de venta de libros).

Relación de uno a uno (1 : 1)



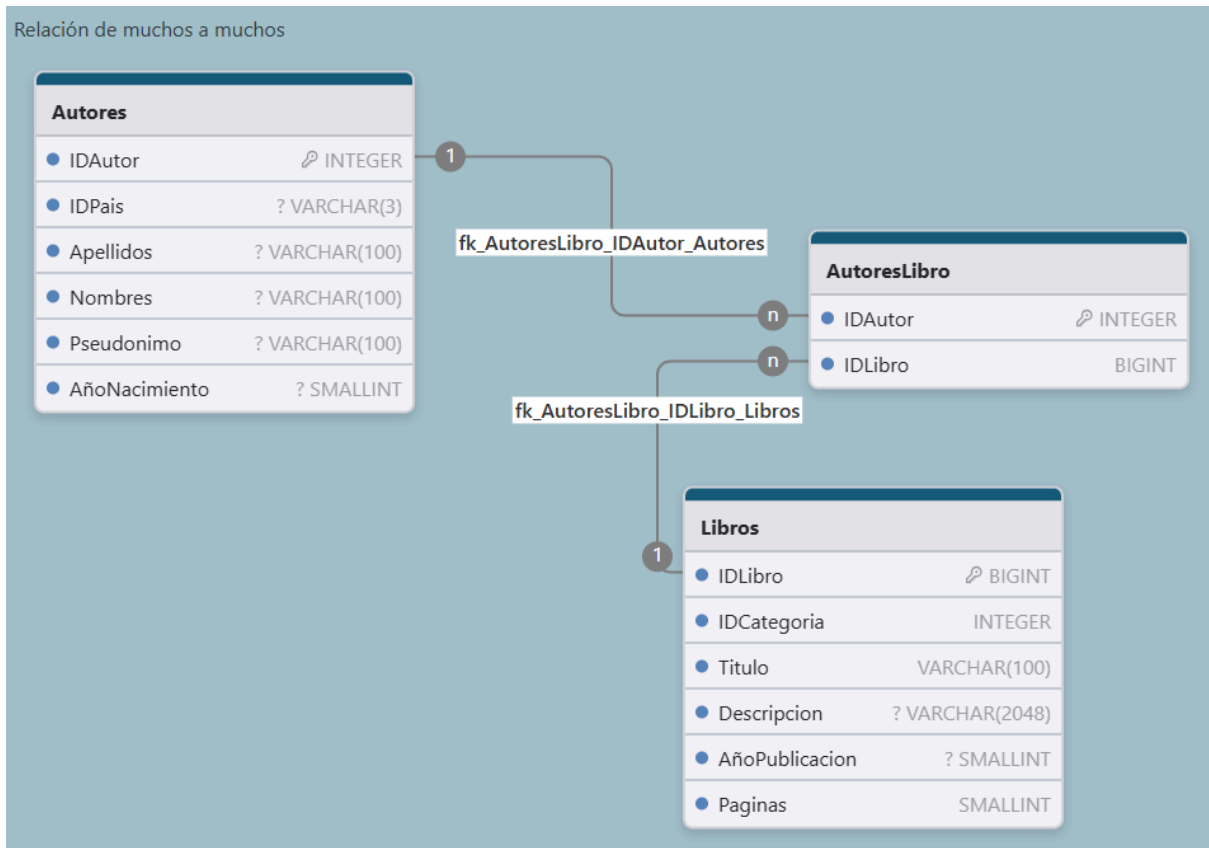
La tabla *Portadas* tiene una relación de uno a uno con la tabla *Libros*, ya que una fila de *Portadas* no puede tener más de una fila coincidente en *Libros*, y viceversa. En términos generales, se crea una relación de uno a uno si las dos columnas relacionadas son claves principales o están definidas como atributos únicos (UNIQUE). En este ejemplo se define que un libro sólo puede tener una portada, y que una portada sólo puede corresponder a un libro.

Relación de uno a muchos (1 : n)



Aquí, decimos que la tabla *Categorias* tiene una relación de uno a varios con la tabla *Libros*, porque una fila de la tabla *Categorias* puede corresponderse con muchas filas en la tabla *Libros*, pero una fila de la tabla *Libros* sólo puede tener correspondencia con una fila de la tabla *Categorias*. Por ejemplo, el `IDCategoria = 1` (Novela) puede repetirse en muchos registros de la tabla de libros, ya que en la tienda puede haber varias existencias de libros con categoría “Novela”. Sin embargo, tal como está planteado este modelo de datos, un libro sólo puede tener una categoría.

Relación de muchos a muchos (n : n)



Finalmente, en la relación de varios a varios tenemos muchas filas de la tabla *Autores* coincidentes con muchas filas de la tabla *Libros*. Para crear este tipo de relación se necesita de una tercera tabla, denominada **tabla de unión** (en este ejemplo es *AutoresLibro*). La clave principal (o primaria) de la tabla de unión es la combinación de las claves principales (o primarias) de las tablas que se relacionan a través de ella (*Autores* y *Libros*), es decir, es una clave compuesta. Dichas claves deben ser claves foráneas en la tabla de unión (*AutoresLibro*).

Creación de bases de datos y tablas por código

Ahora veremos cómo generar bases de datos, tablas y restricciones a través de código T-SQL (*Transact-SQL*). Algunas herramientas (por ejemplo SQL Server Management Studio) permiten realizar estas tareas a través de su entorno gráfico, pero no es lo habitual.

Creación y eliminación de bases de datos

En esta sección trabajaremos con la base de datos vista en las imágenes anteriores.

La sintaxis para crear una base de datos llamada *Libreria* es la siguiente:

```
CREATE DATABASE Libreria  
COLLATE Latin1_General_CI_AI
```

La cláusula COLLATE define la **intercalación** (o collation) de la base de datos. Dicha intercalación se encarga de controlar de qué forma se hará la comparación y ordenamiento de datos en la base de datos, especialmente los que son de tipo char, nchar, varchar, nvarchar, text y ntext. Este paso es opcional y, en caso de no especificar un valor, se asignará la intercalación predeterminada de la instancia de SQL Server.

En el ejemplo estamos asignando la intercalación *Latin1_General_CI_AI* cuyos argumentos son:

- **CollationDesignator** (*Latin1_General*): especifica reglas de ordenación y comparación. En este ejemplo está basado en el idioma inglés y otros idiomas occidentales.
- **CaseSensitivity** (*CI = Case Insensitive*): establece que no se distinguirá mayúsculas de minúsculas. Es decir, si comparamos por ejemplo las cadenas 'BaseDeDatosII' y 'basededatosii", el gestor determinará que son iguales.

Para forzar la distinción entre mayúsculas y minúsculas se debe utilizar el argumento **CS (Case Sensitive)**.

- **AccentSensitivity** (*AI = Accent Insensitive*): establece la no-distinción de acentos. Por ejemplo, una comparación entre 'Programación' y 'Programacion' arrojará como resultado que ambas cadenas son iguales.

Para determinar la distinción de letras con tilde y sin tilde, utilizaremos el argumento **AS (Accent Sensitive)**.

Existen otros parámetros pero el uso que le daremos a esta configuración en la materia no amerita que profundicemos en ella.

Si necesitamos eliminar la base de datos Libreria, previamente tendremos que validar que la base no está siendo utilizada, y luego ejecutar la siguiente instrucción:

```
USE master  
go  
DROP DATABASE Libreria  
go
```

Esta sentencia pone en uso la base de datos del sistema **master** (simplemente a los fines de cambiar a una base que no sea la que queremos eliminar) y luego mediante la sentencia DROP DATABASE eliminamos la base de datos.

La cláusula **go** nos permite ejecutar un conjunto de instrucciones como un proceso por lotes. En el ejemplo anterior, donde realizamos dos operaciones en una misma ejecución, cada bloque de código está delimitado del siguiente por la sentencia *go*.

Creación de tablas

Veremos cómo crear por código la tabla *Libros* vista en el DER (Diagrama Entidad-Relación) de la página 9:

Libros	
IDLibro	BIGINT
IDCategoria	INTEGER
Titulo	VARCHAR(100)
Descripcion	? VARCHAR(2048)
AñoPublicacion	? SMALLINT
Paginas	SMALLINT

La sintaxis podría ser la siguiente:

```
CREATE TABLE Libros(  
  IDLibro BIGINT NOT NULL PRIMARY KEY,  
  IDCategoria INT NOT NULL FOREIGN KEY REFERENCES Categorias  
    (IDCategoria),  
  Titulo VARCHAR(100) NOT NULL,  
  Descripcion VARCHAR(2048) NULL,  
  AñoPublicacion SMALLINT NULL,  
  Paginas SMALLINT NOT NULL CHECK(Paginas > = 1)  
)
```

Aquí podemos notar cómo, luego de escribir el comando para crear la tabla Libros, debemos especificar cada columna con sus características. Cada columna debe tener un nombre y un

tipo de dato. Adicionalmente se pueden incorporar algunos de los elementos que vimos anteriormente:

NULL: las columnas *Descripcion* y *AñoPublicacion* podrían contener ningún dato, es decir, en ellas se admite la ausencia de valor.

NOT NULL: por el contrario, en las columnas *IDLibro*, *IDCategoria*, *Titulo* y *Paginas* no está permitido contener ningún valor. Por defecto, las columnas que forman parte de la clave primaria de una tabla no puede admitir nulos.

PRIMARY KEY: los valores ingresados en las columnas definidas como clave primaria (en este ejemplo, *IDLibro*) no pueden repetirse en la tabla.


FOREIGN KEY: salvo que acepte nulos, no permite ingresar valores en el atributo que no se encuentren reflejados en la tabla donde dicho atributo es clave primaria. En este caso, *IDCategoria* es clave foránea en *Libros* y referencia a *Categorias* (tabla donde el atributo *IDCategoria* es clave primaria). Por lo tanto, no es posible la inserción de un registro en *Libros* cuyo valor en la columna *IDCategoria* no se encuentre en la tabla *Categorias*.

Cabe aclarar que las restricciones de clave primaria y foránea pueden escribirse en la definición de sus correspondientes columnas (es decir, en la misma línea) siempre y cuando no sean claves compuestas (más adelante veremos un ejemplo de estas últimas).

Respecto de las claves foráneas, en su definición se debe indicar a qué tabla y campo hacen referencia. Es obligatorio que dicha tabla y campo existan previamente en la base de datos y que el campo tenga valores válidos (si la columna que lo referencia no admite nulos). Además, la columna que constituye clave foránea debe ser clave primaria en la tabla referenciada.

Columnas **IDENTITY** (o autoincrementales)

Escribiremos un código para crear la tabla *Categorias* mostrada en el DER de página 8:



Categorias	
● IDCategoria	🔑 INTEGER
● Categoria	VARCHAR(50)

```
CREATE TABLE Categorias(  
    IDCategoria INT NOT NULL PRIMARY KEY IDENTITY(1000, 10),  
    Categoria VARCHAR(50) NOT NULL  
)
```

En esta tabla incorporamos una nueva característica que consiste en una columna de tipo **identity** o **autoincremental**. Mediante la sentencia **IDENTITY (inicio, incremento)** hacemos que, en este caso, la columna *IDCategoria* sea autoincremental (o autonumérica). Esto significa que al insertar un registro, se incrementará automáticamente su valor sin necesidad de ser ingresado explícitamente. Es una característica aplicable a las columnas de tipos de datos numéricos.

Los parámetros necesarios para este tipo de atributos son el **valor de inicio** (o **seed**, de *semilla*) que consiste en un número entero a partir del cual comienza la numeración, y el **valor incremental** que también es un número entero y representa el aumento que hay entre un registro y otro.

En el presente ejemplo la numeración de los ID de categorías comenzará en 1000 y se incrementará de 10 en 10.

Columnas **DEFAULT** y **UNIQUE**

Ahora crearemos por código una tabla *Usuarios* que tiene el siguiente diagrama:

Usuarios	
IDUsuario	BIGINT
IDPais	VARCHAR(3)
NombreUsuario	VARCHAR(20)
Mail	VARCHAR(250)
FechaInscripcion	DATE

```
CREATE TABLE Usuarios(  
  IDUsuario BIGINT NOT NULL PRIMARY KEY IDENTITY(1, 1),  
  IDPais VARCHAR(3) NOT NULL FOREIGN KEY REFERENCES Países (IDPais),  
  NombreUsuario VARCHAR(20) NOT NULL UNIQUE,  
  Mail VARCHAR(250) NOT NULL,  
  FechaInscripcion DATE NOT NULL DEFAULT(GETDATE())  
)
```

Aquí nos encontramos con un par de situaciones nuevas:

- El campo *FechaInscripcion* tiene un valor por defecto, definido mediante la palabra reservada **DEFAULT**. El valor predeterminado en este ejemplo está dado por la función **GETDATE()** que devuelve la fecha actual del servidor. Es decir, cuando

insertemos un registro de usuario, su fecha de inscripción será la fecha que posee el servidor al momento de la inserción (a menos que se indique un valor de forma explícita).

- La columna *NombreUsuario* se ha definido como **UNIQUE**. Es decir que, si bien no es la clave primaria de la tabla, se comportará de la misma manera en el sentido de la restricción de duplicidad: no permitirá el ingreso de dos nombres de usuario iguales.

Modificación de tablas

Existen alternativas a la hora de crear una tabla y sus restricciones. Por ejemplo, la tabla *Libros* también podría ser generada de la forma que se muestra a continuación, donde las restricciones de clave primaria y foránea se agregan luego de definir cada una de las columnas:

```
CREATE TABLE Libros(  
    IDLibro BIGINT NOT NULL,  
    IDCategoria INT NOT NULL,  
    Titulo VARCHAR(100) NOT NULL,  
    Descripcion VARCHAR(2048) NULL,  
    AñoPublicacion SMALLINT NULL,  
    Paginas SMALLINT NOT NULL CHECK (Paginas > = 1),  
    PRIMARY KEY (IDLibro),  
    FOREIGN KEY (IDCategoria) REFERENCES Categorias (IDCategoria)  
)
```

Definiendo PK y FK de esta manera, se torna necesario especificar entre paréntesis el nombre de la columna a la cual se le está definiendo la restricción.

Agregar restricciones

Una vez creada una tabla, es posible modificar su estructura mediante la sentencia **ALTER TABLE**. Suponiendo que hemos creado la tabla *Libros* de la siguiente manera:

```
CREATE TABLE Libros(  
    IDLibro BIGINT NOT NULL,  
    IDCategoria INT NOT NULL,  
    Titulo VARCHAR(100) NOT NULL,  
    Descripcion VARCHAR(2048) NULL,  
    AñoPublicacion SMALLINT NULL,  
    Paginas SMALLINT NOT NULL CHECK (Paginas > = 1)  
)  
GO
```

Para incorporar las claves primaria y foránea tendremos que agregar el siguiente código:

```
ALTER TABLE Libros
ADD CONSTRAINT PK_Libros PRIMARY KEY (IDLibro)
GO
```

```
ALTER TABLE Libros
ADD CONSTRAINT FK_Libros_Categorias FOREIGN KEY (IDCategoria) REFERENCES
Categorias (IDCategoria)
```

En este ejemplo ejecutamos un conjunto de instrucciones **DDL (Data Definition Language)** para crear la tabla y agregar sus restricciones posteriormente.

Es importante notar la necesidad de asignar nombres a las claves primaria y foránea respectivamente, y de que los mismos sean lo suficientemente representativos.

Agregar, modificar o quitar columnas

La modificación también puede ser útil cuando necesitamos agregar o quitar una columna, o bien cambiar alguna característica de una ya existente.

Tenemos la tabla *Medios*, que, en el marco de nuestro sistema de tienda de libros, representa los medios de lectura de un libro. Por ejemplo: un libro en papel, un Ebook, un audiolibro, etc. Su estructura actual es la siguiente:

Medios	
IDMedio	TINYINT
Nombre	VARCHAR(50)

Y, una vez diseñada la base de datos, nos piden incorporar una funcionalidad: un listado que muestre la clasificación de los libros disponibles en la tienda según su tipo de medio: si es físico o no.

Entonces necesitaremos agregar un campo llamado *EsFisico* de tipo Bit (booleano) para indicar, en cada registro, si el libro está en formato físico (papel) o virtual (Ebook).

Escribimos el código a continuación para agregar la columna:

```
ALTER TABLE Medios
ADD EsFisico BIT NOT NULL
GO
```

Algunas consideraciones a tener en cuenta:

- Si la tabla que intentamos modificar no está vacía, al asignarle la restricción de no nulidad a la nueva columna, se debe cumplir alguna de estas tres condiciones en su definición:

- Proporcionarle un valor *default*.
- La columna debe ser de tipo *autoincremental*.
- El tipo de dato debe ser *timestamp*.

Caso contrario, el gestor de base de datos no podrá agregar la columna y emitirá un mensaje de error.

- Si la columna a agregar en una tabla con datos está definida como NULL, en caso de no cumplirse ninguna de las tres condiciones anteriores, se le asignará NULL en todos los registros existentes. Cabe aclarar que, de ocurrir la misma situación en el alta de un registro posterior al agregado de la columna, también se rellenará con NULL.

Suponiendo que nuestra base de datos no estaba vacía y finalmente añadimos la columna con la restricción de nulidad:

```
ALTER TABLE Medios
ADD EsFisico BIT NULL
GO
```

Si posteriormente queremos eliminarla, escribiremos lo siguiente:

```
ALTER TABLE Medios
DROP COLUMN EsFisico
GO
```

Es importante destacar que, para eliminar el campo, debemos asegurar previamente la preservación de la integridad referencial, es decir que el mismo no puede estar siendo referenciado en otra tabla (restricción de tipo clave foránea) o bien como default.

Por ejemplo, si al crear la columna la definíamos con una restricción de tipo default:

```
ALTER TABLE Medios
ADD EsFisico BIT NOT NULL DEFAULT (0)
GO
```

Y luego intentamos eliminarla, el gestor de base de datos emitirá un error como el siguiente:

The object 'DF__Medios__EsFisico__6EF57B66' is dependent on column 'EsFisico'.

ALTER TABLE DROP COLUMN EsFisico failed because one or more objects access this column.

Lo que sucede es que la columna *EsFisico* está siendo “utilizada” por la restricción default, que no es ni más ni menos que un objeto dentro de nuestra base de datos. Con lo cual, si aún queremos eliminarla, tendremos que borrar primero la restricción mediante el siguiente código:

```
ALTER TABLE Medios
DROP CONSTRAINT DF__Medios__EsFisico__6EF57B66
GO
```

Y luego sí, estaremos en condiciones de eliminar la columna.

La misma sintaxis se emplea para la eliminación de otro tipo de restricciones, como por ejemplo claves primarias y foráneas, tomando los recaudos vistos anteriormente.

Ahora, a fines de auditar cambios en los registros de la tabla *Usuarios*, solicitan agregar una columna cuyo valor determine si el registro ha cambiado desde la última vez que fue accedido.

El administrador de base de datos decide incorporar la nueva columna de la siguiente manera:

```
ALTER TABLE Usuarios
ADD FechaUltimaActualizacion DATETIME NULL
GO
```

Pero le sugieren que, para un mejor control, el tipo de dato *timestamp* (o *rowversion*) es el más adecuado para dicha columna.

Como no está permitida la modificación directa entre los tipos *datetime* y *rowversion*, el código que tendrá que escribir para modificar el tipo de dato deberá crear la columna con el nuevo tipo de dato, y luego eliminar la columna con el tipo de dato anterior (o viceversa):

```
ALTER TABLE Usuarios
ADD UltimaActualizacion ROWVERSION NULL
GO
ALTER TABLE Usuarios
DROP COLUMN FechaUltimaActualizacion
GO
```

Veamos dos ejemplos más sobre modificaciones en columnas.

Queremos modificar el tipo de dato de la columna *IDAutor* en la tabla *Autores*, ya que nos ha quedado corto:

Autores	
● IDAutor	🔗 INTEGER
● IDPais	? VARCHAR(3)
● Apellidos	? VARCHAR(100)
● Nombres	? VARCHAR(100)
● Pseudonimo	? VARCHAR(100)
● AñoNacimiento	? SMALLINT

Lo convertiremos en BIGINT. Ante todo, debemos recordar que, antes de ejecutar instrucciones DDL, es conveniente resguardar los datos.

Por otro lado, en el DER de página 9 vimos que la tabla *AutoresLibro* referencia a *Autores* (columna *IDAutor*), con lo cual en primer lugar debemos eliminar dicha restricción y luego la PK de la tabla *Autores*, ya que ambos objetos son dependientes de la columna a modificar:

```
ALTER TABLE AutoresLibro
DROP CONSTRAINT FK__AutoresLi__IDAut__44FF419A
GO
ALTER TABLE Autores
DROP CONSTRAINT PK__Autores__5DC53A13F41FF85F
GO
```

Una vez hecho esto, sí podemos proceder a la modificación del tipo de dato:

```
ALTER TABLE Autores
ALTER COLUMN IDAutor BIGINT NOT NULL
GO
```

Agregamos nuevamente la clave primaria eliminada en *Autores*:

```
ALTER TABLE Autores
ADD CONSTRAINT PK_Autores PRIMARY KEY (IDAutor)
GO
```

Pero en *AutoresLibro* necesitaremos trabajar un poco más, ya que al cambiar el tipo de dato de la columna a la que hace referencia IDAutor, que forma parte de su PK, tendremos que cambiar su tipo de dato y rehacer sus claves primaria y foránea:

```
ALTER TABLE AutoresLibro
DROP CONSTRAINT PK__AutoresL__BE3BB4A5AA97B1A3
GO
ALTER TABLE AutoresLibro
ALTER COLUMN IDAutor BIGINT NOT NULL
GO
ALTER TABLE AutoresLibro
ADD CONSTRAINT FK_Autores_Libros FOREIGN KEY(IDAutor) REFERENCES
Autores(IDAutor)
GO
ALTER TABLE AutoresLibro
ADD CONSTRAINT PK_AutoresLibro PRIMARY KEY (IDAutor, IDLibro)
GO
```

Bastante más sencillo sería modificar el tipo de dato de la columna *Categoria* en la tabla *Categorias*, puesto que dicha columna no posee restricciones. Recordemos su estructura anterior:

Categorias	
● IDCategoria	INTEGER
● Categoria	VARCHAR(50)

```
ALTER TABLE Categorias
ALTER COLUMN Categoria NVARCHAR(100) NOT NULL
GO
```

Eliminación de tablas

Finalmente veremos la sintaxis para quitar una tabla de una base de datos. Suponiendo que ya no necesitamos la tabla *Portadas*, ejecutaremos la siguiente instrucción:

```
DROP TABLE Portadas
```

Tal como vimos anteriormente en las modificaciones, es necesario chequear previamente las dependencias de dicha tabla antes de eliminarla, para no dejar a la base de datos en un estado inconsistente.

Claves compuestas

Supongamos que esta tienda de venta de libros dispone de varias sucursales físicas, y en cada una de ellas los empleados son identificados mediante un número de legajo, pero, como entre las distintas sucursales el número de legajo puede repetirse, también se agrega el código de sucursal a su identificación. Por ejemplo:

EMPLEADOS			
Legajo	Sucursal	Apellido	Nombre
1000	San Isidro	Simón	Ángel
2000	San Isidro	Faure	Abel
1000	General Pacheco	Lara Campos	Brian

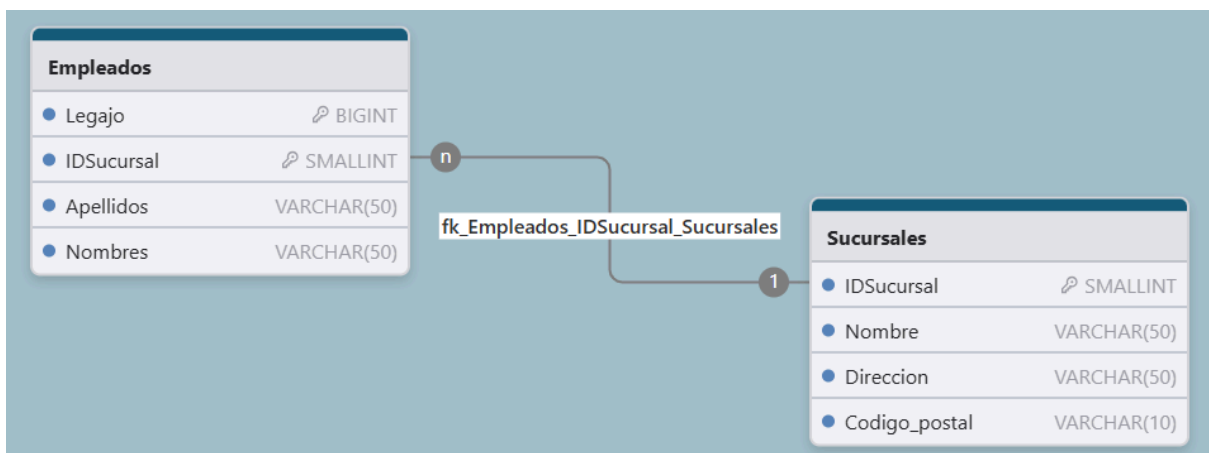
Esto incorpora una complejidad mayor a nuestra estructura de tablas. Ya aprendimos antes acerca del proceso de normalización y la importancia de evitar la redundancia de datos. Por lo tanto, la sucursal debería ser, dentro de *Empleados*, un código que la identifique y que exija integridad referencial con una clave primaria en una tabla que podríamos llamar *Sucursales*:

EMPLEADOS			
Legajo	IDSucursal	Apellido	Nombre
1000	1	Simón	Ángel
2000	1	Faure	Abel
1000	2	Lara Campos	Brian

SUCURSALES			
IDSucursal	Nombre	Dirección	Codigo_postal
1	San Isidro	Av. Centenario 800	1642
2	General Pacheco	Av. H. Yrigoyen 288	1617
3	San Fernando	Av. del Libertador 100	1646

La clave candidata que nos surge pensar automáticamente para la tabla *Empleados* es *Legajo*, pero en verdad no lo es, ya que no asegura por sí sola la representación unívoca de un empleado. Sin embargo, de acuerdo a la lógica de este negocio, si elegimos como clave candidata la combinación de legajo y sucursal, tendremos el problema resuelto: no puede haber más de un empleado con el mismo legajo en la misma sucursal, pero sí pueden existir empleados con el mismo número de legajo en distintas sucursales.

Bajo esta consigna, veamos cómo queda la definición de la tabla *Empleados* con su clave primaria concatenada:



```
CREATE TABLE Empleados(
Legajo BIGINT NOT NULL,
IDSucursal SMALLINT NOT NULL,
Apellidos VARCHAR(50) NOT NULL,
Nombres VARCHAR(50) NOT NULL,
PRIMARY KEY (Legajo, IDSucursal),
FOREIGN KEY (IDSucursal) REFERENCES Sucursales (IDSucursal)
)
GO
```

```
CREATE TABLE Sucursales(
IDSucursal SMALLINT NOT NULL PRIMARY KEY,
Nombre VARCHAR(50) NOT NULL,
Direccion VARCHAR(50) NOT NULL,
Codigo_postal VARCHAR(10) NOT NULL
)
GO
```

Como podemos ver, creamos la tabla *Empleados* definiendo en primer lugar sus columnas y luego sus restricciones de tipo PK y FK. Para la restricción de clave primaria debemos indicar que la misma está conformada por dos columnas (*Legajo* e *IDSucursal*). Además, debemos consignar a la columna *IDSucursal* como clave foránea en la tabla *Sucursales*.

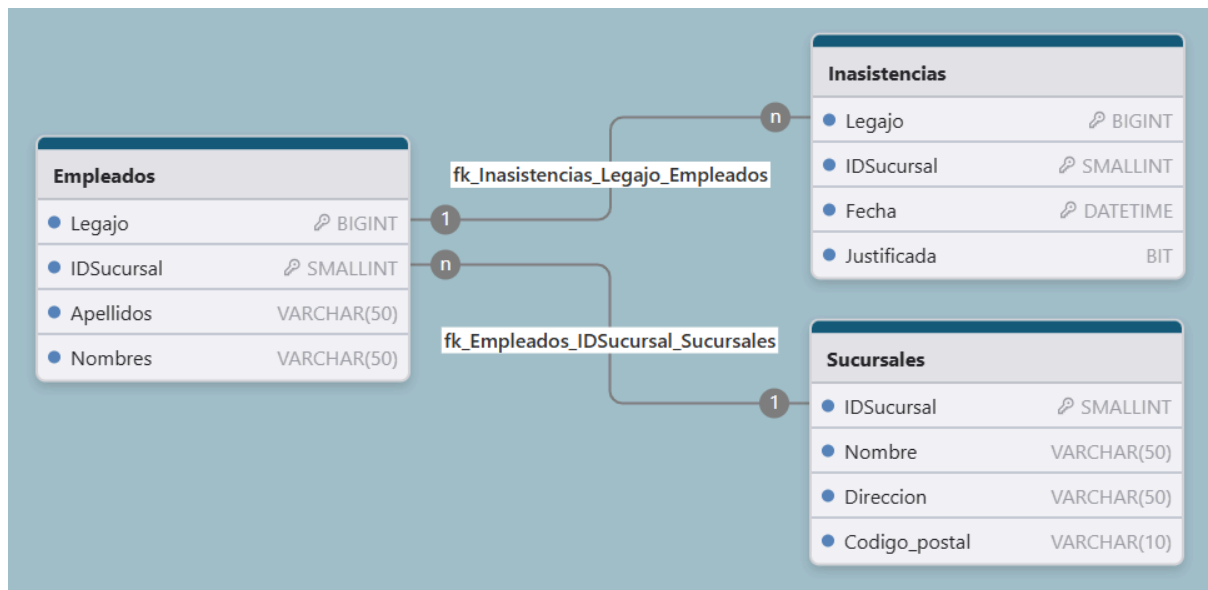
Ahora observemos lo que sucede si la tienda desea registrar también las inasistencias de sus empleados. Para representarlo, debemos crear una tabla *Inasistencias* que almacene en cada registro al empleado, la fecha en la que se ausentó y eventualmente podríamos agregar si la falta fue o no justificada.

Al diseñar la tabla de inasistencias, nuevamente tenemos que pensar en una clave compuesta: el empleado y la fecha en la que faltó. Esto se debe a que un empleado puede faltar en diferentes fechas, y en una fecha pueden faltar distintos empleados, pero sería absurdo registrar que un empleado faltó dos veces en el mismo día.

Lo complejo de esta situación es que, cuando nos referimos al empleado en la tabla de inasistencias, estamos haciendo mención a una clave foránea. Pero, como vimos anteriormente, la clave principal de la tabla de empleados está formada por el legajo del empleado y el ID de la sucursal en la que trabaja. De modo que la clave foránea que realizaremos en *Sucursales* es una clave compuesta que exige integridad referencial sobre la combinación de esas dos columnas.

Así es que, con la definición de estas restricciones, la tabla *Inasistencias* queda formada de la siguiente manera:

```
CREATE TABLE Inasistencias(
Legajo BIGINT NOT NULL,
IDSucursal SMALLINT NOT NULL,
Fecha DATETIME NOT NULL,
Justificada BIT NOT NULL DEFAULT (0),
PRIMARY KEY (Legajo, IDSucursal, Fecha),
FOREIGN KEY (Legajo, IDSucursal) REFERENCES Empleados (Legajo, IDSucursal)
)
GO
```



Podemos observar una clave principal formada por tres columnas (*Legajo*, *IDSucursal* y *Fecha*) y una clave foránea conformada por dos columnas (*Legajo* e *IDSucursal*). Esta clave foránea exigirá integridad referencial por una ocurrencia de ambos valores. Es decir, deberá existir un registro en *Empleados* que sea coincidente en la misma combinación de los dos valores (legajo y el ID de la sucursal) para que se permita el ingreso de una inasistencia.

La especificación DEFAULT (0) en la columna booleana *Justificada*, tendrá la función de establecer el valor por defecto 0 (o *False*) en caso de insertar un registro de inasistencia sin dicho valor suministrado.