# TEMA 4: Diseño Físico

Módulo

Bases de Datos

para los ciclos

Desarrollo de Aplicaciones Web

Desarrollo de Aplicaciones Multiplataforma

## *4.1 Introducción*

A modo de repaso recordaremos que el diseño de una base de datos pasa por tres fases:

- ***Modelo Conceptual***: Consiste en expresar el trozo de mundo que queremos controlar de una forma objetiva e inteligible para todo el mundo, de modo que se pueda utilizar para entenderse con el cliente que conoce ese trozo de mundo. Nosotros hemos usado para esta fase el ***Diagrama Entidad/Relación*** en su notación clásica.
- ***Modelo Lógico***: Consiste en expresar ese trozo de mundo ya modelado, de una forma técnica directamente traducible a un lenguaje informático. Existen varias opciones para esta fase: ***modelo relacional***, ***modelo jerárquico*** y ***modelo distribuido***. Nosotros hemos usado el primero de ellos.
- ***Modelo Físico***: Se trata de implementar nuestra base de datos en un SGBD que nos permita insertar los datos y manipularlos con ayuda de un ordenador. Para esta fase el lenguaje más extendido es ***SQL*** (Standard Query Language, Lenguaje estándar de consultas).

  El lenguaje ***SQL*** es sólo una definición teórica que indica cómo debería de crearse una tabla, cómo definir sus claves primarias o ajenas, cómo se debería escribir una consulta para extraer datos, etc. Para trabajar con SQL tendremos que elegir alguna de las muchas implementaciones reales que existen (***MySQL***, ***MariaDB***, ***Oracle***, ***DB2***, ***SQLite***, etc).

  En esta primera aproximación vamos a utilizar ***MySQL*** que es una implementación libre y gratuita de SQL. En los temas siguientes trabajaremos también con la versión de ***Oracle***, que es propietaria.

## *4.2 Instalación de MySQL*

### *4.2.1 Instalación MySQL en Linux*

Para instalar ***MySQL*** en una distribución de ***Linux*** bastará con instalar el paquete ***mysql-server***, que normalmente se encuentra en los repositorios de la propia distribución. Esto lo podemos hacer desde el ***Synaptic*** o bien, desde la línea de comandos:

```
sudo apt-get install mysql-server
```

La instalación inicial sólo tendrá un usuario llamado ***root*** (no confundir con el usuario ***root*** del sistema) y se te solicitará una contraseña (que hay que introducir dos veces) para ese usuario.

***Nota***: La contraseña del usuario ***root*** de ***MySQL*** en un entorno de producción deberá ser una contraseña segura y conocida solamente por el administrador de la BD. Sin embargo, durante las clases vamos a usar todos la misma contraseña: ***root***.

***MySQL*** se ejecutará al inicio de cada sesión. No obstante, si en algún momento necesitas iniciarlo o detenerlo tendrás que ejecutar las siguientes instrucciones:

- Para iniciarlo:

```
/etc/init.d/mysql start
```

- Para detenerlo:

```
/etc/init.d/mysql stop
```

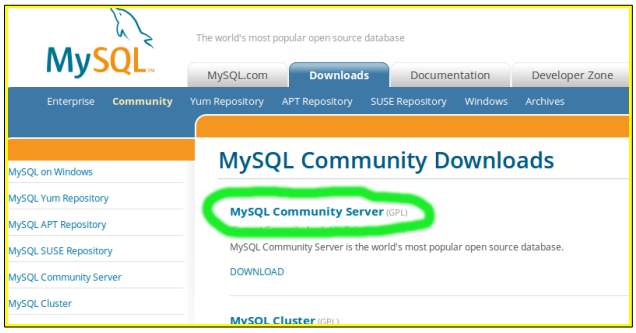
- Para reiniciarlo:

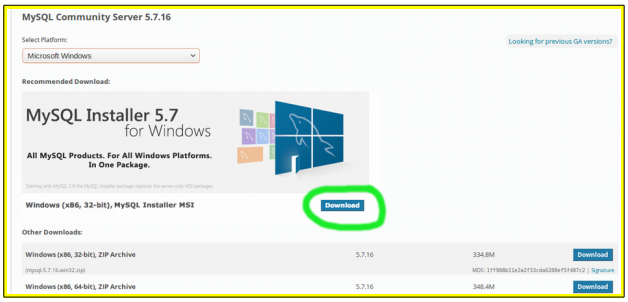
```
/etc/init.d/mysql restart
```

***Nota***: ***Mariadb*** es igual a ***MySQL***, pero no pide contraseña al instalar y sólo es posible iniciar ***mariadb*** como ***root*** desde el usuario ***root*** del sistema. El paquete a descargar es: ***mariadb-server***.

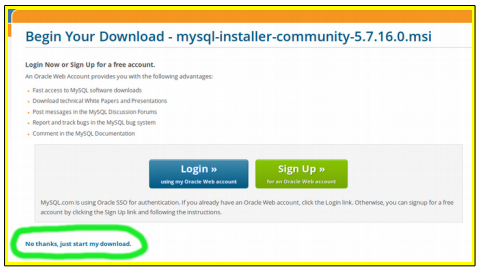
### *4.2.2 Instalación MySQL en Windows*

La descarga la realizamos desde la web http://dev.mysql.com/downloads/ eligiendo el ***instalador .msi para Windows de 64 bits***. Para ello sigue la secuencia de imágenes:



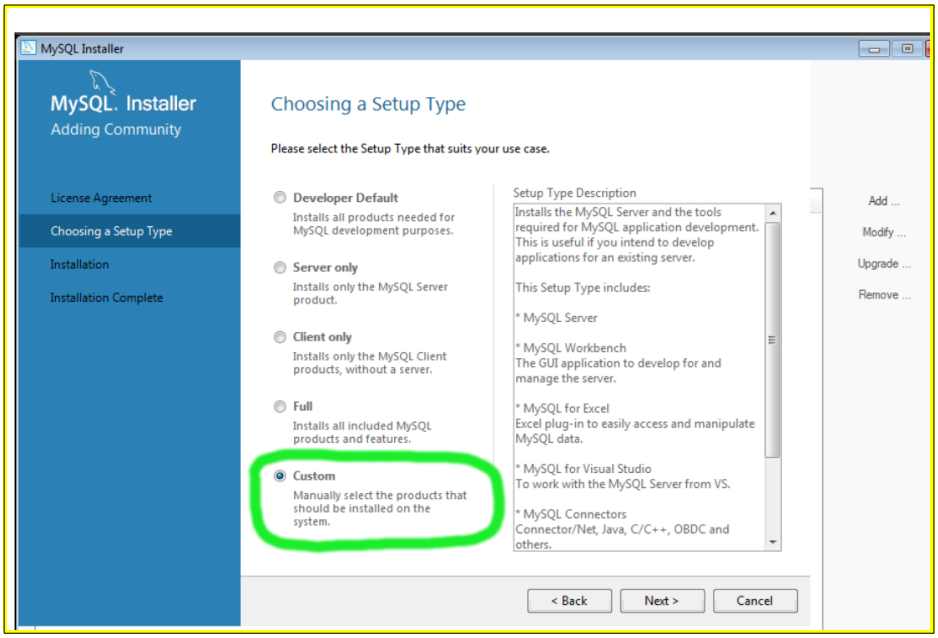




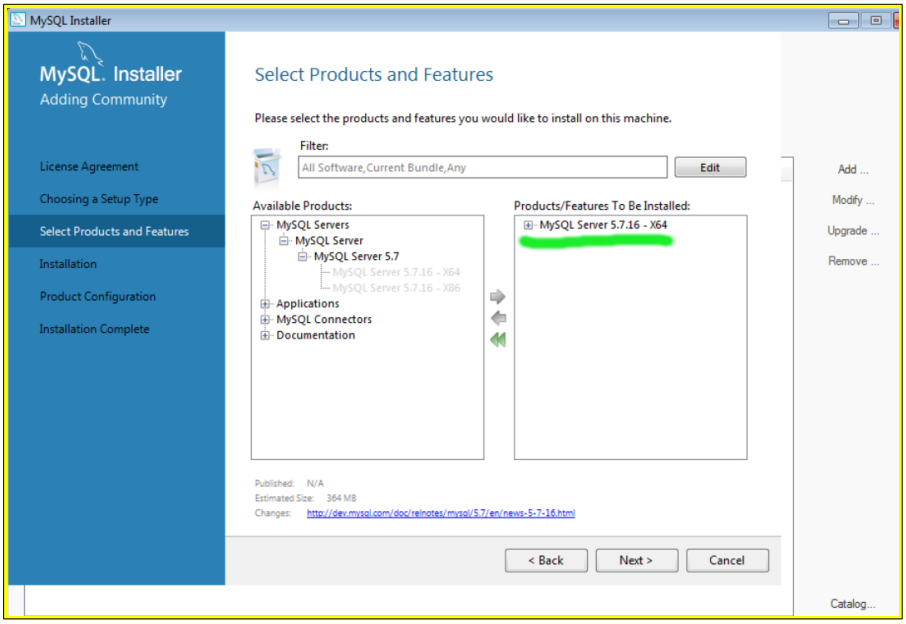

***Nota***: Usando la ***huella digital*** que aparece en la web podrás comprobar que la descarga se ha realizado correctamente. Si te acostumbras a hacerlo evitarás muchos problemas.

Para instalar la aplicación sólo tienes que ejecutar el instalador ***.msi*** que te has descargado y seguir las siguientes instrucciones:

- Si parece que la instalación se paraliza probablemente haya una ventana escondida esperando que autorices la instalación.
- Acepta los términos.
- Elige la instalación personalizada (***Custom***), como en la siguiente imagen:




- Asegúrate de que sólo instalas el ***servidor de MySQL para 64 bits***. No necesitamos el resto de aplicaciones ni conectores.




- Continúa aceptando todas las opciones por defecto, salvo la contraseña del usuario ***root***, que debería ser ***root***.

***Nota***: La contraseña del usuario ***root*** de ***MySQL*** en un entorno de producción deberá ser una contraseña segura y conocida solamente por el administrador de la BD. Sin embargo, durante las clases vamos a usar todos la misma contraseña: ***root***.

Una vez terminada la instalación, desde un terminal de comandos, desplázate hasta la carpeta:

```
C:\Program Files\MySQL\MySQL Server 5.7\bin
```

Desde esa carpeta podrás ejecutar ***mysql***.

***Nota***: Para poder ejecutar ***mysql*** desde cualquier otro sitio tendrás que añadir la carpeta a la ***variable de entorno PATH***. Por ejemplo, en ***Windows 7*** debes ir a:

```
Inicio/Panel de Control/Sistema y Seguridad/Sistema/Cambiar configuración/Opciones avanzadas/Variables de entorno
```

editar la variable ***PATH*** y añadirle al final:

```
:C:\Program Files\MySQL\MySQL Server 5.7\bin
```

### 4.2.3 Primeros pasos

#### Entrar en MySQL

```
mysql -u nombreUsuario -p
```

Por ejemplo:

```
mysql -u root -p
```

Debes tener en cuenta lo siguiente:

- El prompt habitual de MySQL es:

```
mysql>
```

y significa que está a la espera de un comando, pero existen otros:

```
->
```

significa que espera la finalización de un comando incompleto.

```
'>
```

significa que espera que se cierre una comilla simple.

```
">
```

significa que espera que se cierren unas comillas dobles.

- MySQL no es case sensitive para palabras reservadas. Esto significa que se pueden escribir con cualquier combinación de mayúsculas/minúsculas.
- MySQL si es case sensitive para identificadores de campo, tabla, índice, etc, en sistemas derivados de UNIX. Por tanto, siempre se deberían escribir con la combinación de mayúsculas/minúsculas correctas.
- Todos los comandos MySQL terminan con punto y coma (**;**).
- Para anular un comando escribimos ***c** o la combinación de teclas CTRL - C .
- Algunos comandos son:

```
select version();
```

Muestra la versión de MySQL.

```
select current_date();
```

Muestra la fecha actual.

```
select now();
```

Muestra fecha y hora actual.

```
select user();
```

Muestra el usuario actual.

```
select user from mysql.user;
```

Muestra todos los usuarios de MySQL.

```
show databases;
```

Muestra un listado de las bases de datos. En la instalación inicial habrá al menos 4 bases de datos que pertenecen al sistema y que no debes manipular de momento: information_schema, mysql, performance_schema y sys.

```
use nombreBD;
```

Activa la base de datos ***nombreBD***. Es lo que tenemos que hacer antes de manipular una base de datos.

```
show tables;
```

Muestra todas las tablas de la base de datos activa.

```
describe nombreTabla;
```

Muestra una descripción de la tabla ***nombreTabla*** que incluye los nombres de los campos, su tipo, si forman parte de la clave primaria, etc.

- En http://dev.mysql.com/doc/refman/5.7/en/ se encuentra el manual oficial.

### SALIR DE MYSQL

```
exit
```

o bien:

```
quit
```

### CREAR/ELIMINAR BASES DE DATOS

```
CREATE DATABASE nombreBD;
```

Crea una base de datos con el nombre ***nombreBD***.

```
CREATE DATABASE IF NOT EXISTS nombreBD;
```

Crea una base de datos con el nombre ***nombreBD***, pero solo en el caso de que no exista previamente. En realidad nunca permite crear una base de datos con el nombre de otra ya existente, pero al incluir ***IF NOT EXISTS*** evita el mensaje de error.

```
DROP DATABASE nombreBD;
```

Para eliminar la base de datos de nombre ***nombreBD***;

***Nota***: Cada vez que creamos una base de datos con MySQL se genera una carpeta con el mismo nombre en ***/var/lib/mysql***, que contiene todos los ficheros que conforman la BD. Esta ruta puede variar en función de la instalación que hayamos hecho.

En el caso de ***Windows 7***, si ***C:\Program Files\MySQL\MySQL Server 5.7\bin*** es la ruta de instalación, las bases de datos estarían en: ***C:\ProgramData\MySQL\MySQL Server 5.7\Data***. Ten en cuenta que la carpeta ***C:\ProgramData*** está oculta.

***CREAR USUARIOS***

Al igual que en el sistema operativo, el usuario ***root*** tiene todos los privilegios, pero no es una buena práctica operar siempre con él. Para crear un usuario:

```
CREATE USER nombreUsuario IDENTIFIED BY 'contraseña';
```

Un usuario recién creado no tiene ningún permiso. Para permitir a un usuario acceder a una base de datos:

```
GRANT ALL PRIVILEGES ON nombreBD.* TO 'nombreUsuario';
```

Para cambiar la contraseña de un usuario:

```
SET PASSWORD FOR nombreUsuario = password('contraseña');
```

## 4.3 El Lenguaje DDL de SQL

***DDL*** (Data Definition Language, Lenguaje de definición de datos) es la parte de ***SQL*** que permite crear tablas y definirlas.

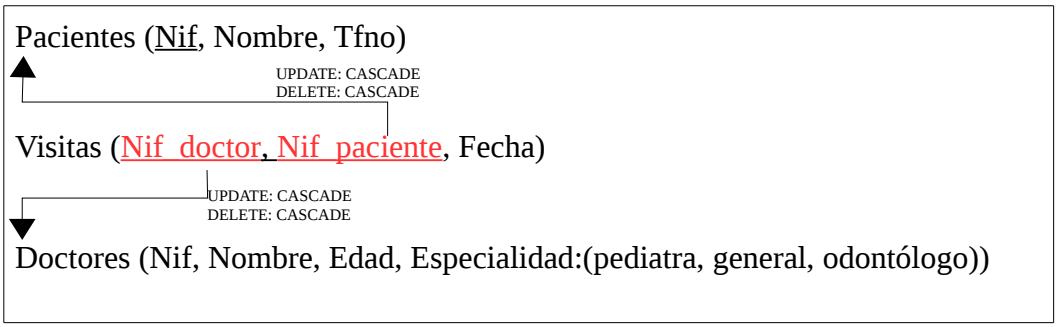
### 4.3.1 Crear una tabla

```
CREATE TABLE nombreTabla (
    nombre tipo,
    ….
    nombre tipo,
    PRIMARY KEY (nombreCampo1, …, nombreCampoN),
    FOREIGN KEY (campoClaveAjena) REFERENCES nombreTabla(claveTabla) ON UPDATE actual ON DELETE borrado
);
```

Observa los siguientes detalles:

- Todo lo que está en color blanco se escribe tal cual. Lo que está en amarillo tendrás que sustituirlo en cada caso.
- Se puede escribir todo en una línea, pero se suele usar la estructura que ves para darle claridad.
- La última línea antes del paréntesis de cierre NO termina en coma (**,**).
- La línea "***FOREIGN KEY***" define una clave ajena. Esto, naturalmente, sólo es necesario en las tablas que tengan claves ajenas.

***Ejemplo: código SQL para un grafo relacional.***




```
CREATE TABLE Pacientes (
    Nif char(9),
    Nombre varchar(40),
    Tfno char(9),
    PRIMARY KEY (Nif)
);

CREATE TABLE Doctores (
    Nif char(9),
    Nombre varchar(40),
    Edad integer,
    Especialidad enum ('pediatra', 'general', 'odontólogo'),
    PRIMARY KEY (Nif)
);

CREATE TABLE Visitas (
    Nif_doctor char(9),
    Nif_paciente char(9),
    Fecha date,
    PRIMARY KEY (Nif_doctor, Nif_paciente),
    FOREIGN KEY (Nif_doctor) REFERENCES Doctores(Nif) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (Nif_paciente) REFERENCES Pacientes(Nif) ON UPDATE CASCADE ON DELETE CASCADE
);
```

## *TIPOS DE DATOS*

Los tipos de datos más habituales son:

- ***char(n)***: Para cadenas de caracteres de hasta ***n*** caracteres. Independientemente del dato que alojen, ocuparán el espacio de ***n*** caracteres.
- ***varchar(n)***: Para cadenas de caracteres de hasta n caracteres. Ocuparán sólo el espacio necesario para alojar el dato.
- ***integer***: Para valores numéricos enteros.
- ***float***, ***double***: Para valores numéricos reales. La parte decimal se separa con un punto (.).

- ***date***: Para fechas. El formato de fecha es ***'aaaa-mm-dd'***. Fíjate en el orden, la separación con guiones (-) y que va entrecomillado.
- ***boolean***, ***bool***: Para valores lógicos. En realidad se almacena como un entero con valor ***0*** para ***false*** y valor ***1*** para ***true***, pero permite asignar y comparar con ***true*** y ***false***.
- ***text***: Para cadenas de caracteres de larga extensión. Por ejemplo diagnóstico de un paciente, comentario de un gol, etc.
- ***enum***: Permite crear tipos a medida que contienen un grupo de valores concretos, por ejemplo el campo ***color*** de una ficha de parchís se podría definir así:

```
color enum('rojo', 'verde', 'amarillo', 'azul')
```

Existen otros muchos tipos de datos que iremos introduciendo más adelante.

***Nota***: Recuerda que algunos datos como los números de teléfono, los códigos postales o DNI sin letra ***pueden parecer datos numéricos, pero en realidad no lo son***. Sólo se tratarán como datos numéricos aquellos con los que pueda ser necesario hacer operaciones matemáticas.

### *DEFINIR CLAVES PRIMARIAS*

Hay tres formas de definir una clave primaria:

```
CREATE TABLE Pacientes (
    Nif char(9) PRIMARY KEY,
    Nombre varchar(40)
);
```

Esta primera sólo admite claves primarias atómicas. Si la clave está compuesta de 2 o más campos será necesario usar la siguiente forma:

```
CREATE TABLE Pacientes (
    Nif char(9),
    Nombre varchar(40)
    PRIMARY KEY (Nif, Nombre)
);
```

Si, además queremos establecer un nombre para la clave primaria usaremos esta tercera forma:

```
CREATE TABLE Pacientes (
    Nif char(9),
    Nombre varchar(40),
    CONSTRAINT pk_Pacientes PRIMARY KEY (Nif)
);
```

Poner nombre a las claves será de mucha utilidad cuando recibamos mensajes de error de ***MySQL***. En los ejercicios propuestos en este módulo siempre pondremos un nombre a las claves primarias que será:

```
pk_NombreTabla
```

### *DEFINIR CLAVES AJENAS*

También hay varias formas de establecer las claves ajenas:

```
CREATE TABLE Visitas (
  Nif_doctor char(9),
  Nif_paciente char(9) REFERENCES Pacientes(Nif) ON UPDATE CASCADE ON DELETE CASCADE,
  CONSTRAINT pk_Visitas PRIMARY KEY (Nif_doctor, Nif_paciente),
);
```

Observa que en la propia definición del campo ***Nif_paciente*** se establece que hace referencia la tabla ***Pacientes*** cuya clave primaria es ***Nif*** y especifica como deben tratarse las actualizaciones y los borrados.

De la misma forma que con las claves primarias, esta versión no ofrece la posibilidad de usar claves ajenas compuestas y tampoco permite establecer un nombre para las claves ajenas. En los ejercicios propuestos en este módulo siempre pondremos nombre a las claves ajenas que será:

```
fk_estaTabla_tablaReferenciada
```

Y por tanto el ejemplo anterior quedaría así:

```
CREATE TABLE Visitas (
    Nif_doctor char(9),
    Nif_paciente char(9),
    CONSTRAINT pk_Visitas PRIMARY KEY (Nif_doctor, Nif_paciente),
    CONSTRAINT fk_Visitas_Pacientes FOREIGN KEY (Nif_paciente)
          REFERENCES Pacientes(Nif)
          ON UPDATE CASCADE ON DELETE CASCADE
);
```

***MUY IMPORTANTE***: Para poder establecer una clave ajena es necesario que su tipo de datos sea exactamente el mismo que el de la clave de la tabla referenciada.

### *CARACTERÍSTICAS DE UN CAMPO*

Cuando se define un campo en la creación de una tabla se pueden añadir condiciones o características que deben cumplir los valores de ese campo. Algunos de ellos ya los conocemos, como la declaración de clave primaria o clave principal.

- ***NOT NULL*** o ***NULL***: Indica que ese campo no admite valores nulos o si los admite respectivamente. Si no se especifica, admitirá nulos.
- ***DEFAULT valor***: Establece un valor por defecto para el campo, supuesto que al añadir un registro no se le asigne un valor.
- ***UNIQUE KEY*** o ***PRIMARY KEY***: Indica que el campo no puede tomar el mismo valor en registros distintos. En el caso de ***PRIMARY KEY*** establece además que este campo será la clave primaria (lo que lleva implícito que es ***UNIQUE KEY***).
- ***AUTO_INCREMENT***: Indica que el campo tomará valores correlativos automáticamente para cada nuevo registro. El tipo del campo tendrá que ser alguno de los tipos enteros y es conveniente usar un tipo sin signo (***UNSIGNED***) para aumentar el rango de valores disponible.

***Ejemplo: Características de un campo***

```
CREATE TABLE Clientes (
    Codigo SMALLINT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    Dni char(9) UNIQUE KEY NOT NULL,
    Nombre varchar(40) NOT NULL,
    Moroso boolean DEFAULT false
);
```

- El campo ***Codigo*** es la clave primaria, que se incrementa automáticamente para cada nuevo registro. Fíjate que ha sido definido del tipo ***SMALLINT*** (pequeño entero, 65.536 valores distintos) y ***UNSIGNED*** para que sólo use valores positivos (desde 0 hasta 65.536) en lugar de positivos y negativos (desde -32.768 hasta 32.767).
- El campo ***Dni*** es una clave candidata (porque no se repiten sus valores) y además no admitirá valores nulos.
- El campo ***Nombre*** no admite valores nulos
- El campo ***Moroso*** toma el valor por defecto ***false***, salvo que explícitamente se le asigne ***true***.

### *4.3.2 Scripts*

Trabajar directamente en la consola de ***MySQL*** puede ser incómodo, por eso tenemos otra opción que consiste en escribir el guión o script de órdenes en un fichero de texto con la extensión ***.sql*** y posteriormente lanzar la ejecución completa con la orden (desde el terminal del sistema operativo):

```
mysql -u usuario -p < fichero.sql
```

O bien con la orden (desde el terminal de ***MySQL***):

```
source fichero.sql
```

En los scripts se pueden incluir comentarios precedidos del símbolo almohadilla (#) o de dos guiones y un espacio (-- ), además será necesario comenzar indicando la BD con la que queremos trabajar y tener en cuenta que hay que ordenar el borrado de una tabla antes de crearla (porque quizá ya fue creada en un intento anterior de ejecución del script antes de corregir los últimos errores).

***Ejemplo: Script SQL***

```
#Ejemplo de script SQL

-- es necesario indicar la BD con la que queremos trabajar
USE empresa;

-- habrá que borrar la tabla si ya existía
DROP TABLE IF EXISTS Clientes;

CREATE TABLE Clientes (
    Dni char(9),
    Nombre varchar(40),
    CONSTRAINT pk_clientes PRIMARY KEY (Dni)
);
```

***Nota***: Las tablas no se pueden crear en cualquier orden. Si una tabla tiene una clave ajena que apunta a otra, la tabla referenciada tendrá que crearse antes. De la misma forma, para borrar una tabla es necesario que no esté referenciada. Por tanto, se borran en orden contrario al orden en que se crean.

## MODIFICAR LAS TABLAS

En ocasiones es necesario modificar una tabla (añadir un campo, modificar un tipo, añadir o borrar una clave ajena). Podríamos pensar que basta con borrar la tabla entera y volver a crearla, pero eso no siempre es posible. Por ejemplo, en una base de datos en producción, borrar la tabla significa borrar todos sus datos.

Otro ejemplo en el que se necesita modificar una tabla es en el caso de que existan dos tablas que se referencian mutuamente. Como ya sabes, no es posible apuntar a una tabla desde una clave ajena si no se ha creado previamente esa tabla. Si dos tablas se referencian mutuamente no podemos completar la clave ajena de la primera tabla, puesto que la segunda aún no existe.

Para este último caso deberemos crear una primera tabla sin la clave ajena, a continuación crear la segunda tabla y finalmente, modificar la primera tabla para referenciar a la primera.

***Ejemplo: uso de ALTER para crear tablas con referencias cruzadas***

```
#Creamos la tabla Alumnos sin clave ajena a Cursos porque aún no se ha
#creado esa tabla. Añadimos la clave ajena al final con ALTER TABLE

CREATE TABLE Alumnos (
    Codigo char(5),
    Nombre varchar(40),
    Curso char(5),
    CONSTRAINT pk_alumnos PRIMARY KEY (Codigo)
);

CREATE TABLE Cursos (
    Codigo char(5),
    Delegado char(5),
    CONSTRAINT pk_grupos PRIMARY KEY (Codigo),
    CONSTRAINT fk_cursosAlumnos FOREIGN KEY (Delegado)
          REFERENCES Alumnos(Codigo)
          ON UPDATE CASCADE ON DELETE RESTRICT
);

ALTER TABLE Alumnos
    ADD CONSTRAINT fk_alumnosCursos FOREIGN KEY (Curso)
                REFERENCES Cursos(Codigo)
                ON UPDATE CASCADE ON DELETE RESTRICT;
```

Con ***ALTER*** podrás añadir, modificar o eliminar campos, claves primarias y ajenas de una tabla. Puedes buscar en la página oficial de ***MySQL*** todas las posibilidades que ofrece y consultar la sintaxis.

***Ejemplo: uso de ALTER para borrar tablas con referencias cruzadas***

Para eliminar las tablas del ejemplo anterior tendremos el mismo problema. Las referencias cruzadas no nos dejan eliminar ninguna de las dos, por eso, tendremos que usar ***ALTER*** para eliminar primero una de las claves ajenas:

```
ALTER TABLE Alumnos DROP FOREIGN KEY fk_alumnosCursos;
DROP TABLE IF EXISTS Cursos;
DROP TABLE IF EXISTS Alumnos;
```