Unidad 4 : Diseño Físico relacional: DDL y DCL

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Fecha	Versión	Descripción
30/10/2022	1.0.0	Versión completa

Unidad 4 : Diseño Físico relacional: DDL y DCL

1. Introducción

En la fase de análisis se realiza la especificación de requisitos de nuestro sistema. A partir de dicha especificación de requisitos, en la unidad 2, aprendimos a realizar el diseño conceptual de una BD mediante el Modelo E/R y el Modelo E/R extendido respectivamente.

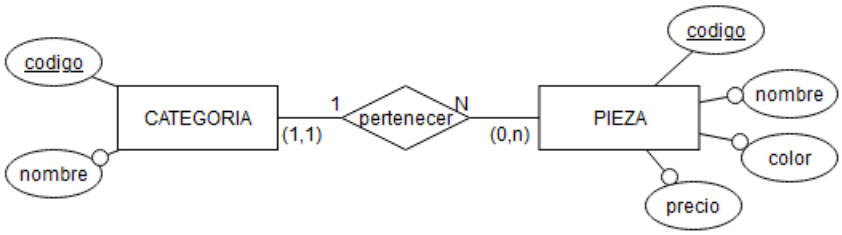
En la unidad anterior vimos como traducir nuestro MER al esquema el modelo relacional y aprendimos a comprobar que dicho modelo estaba normalizado (si hemos realizado un buen diseño inicial, esta parte no suele ser necesaria).

Siguiendo con el proceso de desarrollo, lo que debemos hacer ahora es pasar al diseño físico de la BD. Es decir, implementar la Base de Datos, haciendo uso del lenguaje más extendido para la definición y manipulación de datos en SGBDR: **SQL**, concretamente el DDL (*Definition Data Language*).

En nuestro caso hemos optado por usar el SGBD MySQL, uno de los más populares y usados del mundo. Si bien es cierto que tiene una competencia fuerte, sobre todo con SQLite, Oracle y Microsoft SQL Server, entre otros, MySQL tiene una serie de ventajas que la hacen única:

- Es una base de datos gratuita (de código abierto).
- Es muy fácil de usar.
- Ofrece un alto nivel de seguridad por lo que funciona muy bien para un uso más profesional.
- Es una base de datos muy rápida (aunque no es la base de datos más rápida del mercado).
- No consume apenas recursos (CPU o RAM) de nuestro ordenado.
- Una de sus mayores virtudes es que MySQL es compatible tanto con Windows como con Linux y UNIX.

La base de datos que vamos a crear tiene el siguiente Modelo Entidad-Relación y su correspondiente Esquema Relacional:



REx1: El precio no puede ser negativo

```
CATEGORIA (codigo, nombre)
    PK = codigo
    VNN = nombre
PIEZA (codigo, nombre, color, precio, cod_cat)
    PK = codigo
    FK = cod_cat --> CATEGORIA (codigo)
    VNN = nombre, color, precio, cod_cat
```

REx1: El precio no puede ser negativo

2. El Lenguaje SQL

El nacimiento del **lenguaje** SQL data de 1970 cuando E. F. Codd publica su libro: «*Un modelo de datos relacional para grandes bancos de datos compartidos*». Ese libro dictaría las directrices de las bases de datos relacionales. Apenas dos años después IBM (para quien trabajaba Codd) utiliza las directrices de Codd para crear el **Standard English Query Language** (Lenguaje Estándar Inglés para Consultas) al que se le llamó **SEQUEL**. Más adelante se le asignaron las siglas **SQL** (**Standard Query Language**, lenguaje estándar de consulta) aunque en inglés se siguen pronunciando "*secuel*". En español se pronuncia "*esecuele*".

Este lenguaje se **convirtió en estándar en 1986** por la ANSI (*American National Standards Institute*), y posteriormente por ISO (*International Organization for Standardization*), en 1987.

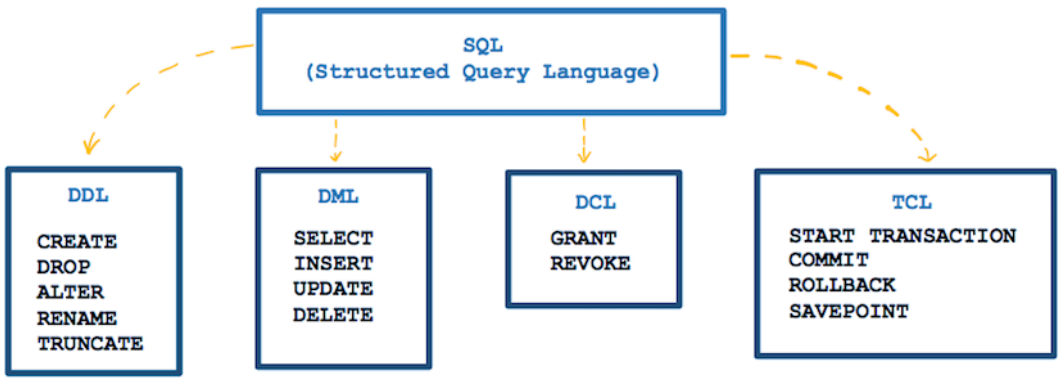
No obstante, existen **numerosos dialectos de SQL** implementados por diversas alternativas de motores de bases de datos. Estos dialectos pueden no cubrir en estándar completo o tener variaciones leves como el tratamiento de mayúsculas y minúsculas, los índices, etc.

Los motivos de la falta de compromiso con el el estándar SQL son variados, desde que el propio estándar no cubre todas las posibilidades del lenguaje en todos los escenarios hasta que las tecnologías de bases de datos son tan variadas que es complejo adaptarse a unas reglas únicas.

Sucesivas revisiones han sido publicadas a lo largo del tiempo, cada dos o tres años. Algunas de las últimas en 2016 y 2019.

El lenguaje SQL se divide en cuatro sublenguajes:

- **Lenguaje de definición de datos (DDL)**: proporciona órdenes para definir, modificar o eliminar los distintos objetos de la base de datos (tablas, vistas, índices...). Lo forman las instrucciones **CREATE**, **DROP** y **ALTER**.
- **Lenguaje de Manipulación de Datos (DML)**: proporciona órdenes para insertar, suprimir y modificar registros o filas de las tablas. También contempla la realización de consultas sobre la BD. Lo forman las instrucciones **SELECT, INSERT**, **DELETE** y **UPDATE**.
- **Lenguaje de Control de Datos (DCL)**: permite establecer derechos de acceso de los usuarios sobre los distintos objetos de la base de datos. Lo forman las instrucciones **GRANT** y **REVOKE**.
- **Lenguaje para el control de transacciones (TCL)**: son comandos que permiten especificar el comienzo y el final de una transacción. Lo forman las instrucciones **START TRANSACTION, ROLLBACK**, **COMMIT** y **SAVEPOINT**.



En esta unidad nos centraremos básicamente en el **DDL** y **DCL**.

2.1. Lenguaje de Definición de Datos (DDL)

El **DDL** (*Data Definition Language*) o **Lenguaje de Definción de Datos** es la parte de SQL dedicada a la definición de los datos. Las sentencias **DDL** son las siguientes:

- `CREATE` : se utiliza para crear objetos como bases de datos, tablas, vistas, índices, *triggers* y procedimientos almacenados.
- `DROP` : se utiliza para eliminar los objetos de la base de datos.
- `ALTER` : se utiliza para modificar los objetos de la base de datos.
- `SHOW` : se utiliza para consultar los objetos de la base de datos.

Otras sentencias de utilidad son:

- `USE` : se utiliza para indicar la base de datos con la que queremos trabajar.
- `DESCRIBE` : se utiliza para mostrar información sobre la estructura de una tabla.
- `SHOW` : se utiliza para consultar los objetos de una base de datos

2.2. Lenguaje de Control de Datos (DCL)

El **DCL** (*Data Control Language*) o **Lenguaje de Control de Datos** es la parte de SQL que permite al administrador controlar el acceso a usuarios mediante la asignación de permisos o roles para realizar determinadas tareas. Las sentencias **DDL** más importantes son las siguientes:

- `GRANT` : se utiliza para otorgar permisos a los usuarios.
- `REVOKE` : se utiliza para eliminar los permisos que previamente se han concedido.

2.3. Criterios de notación y Normas de escritura

La notación utilizada para la especificación de los comandos de SQL es la siguiente:

- Palabras clave de la sintaxis SQL en MAYÚSCULAS.
- Los corchetes [ ] indican opcionalidad.
- Las llaves { } delimitan alternativas separadas por | de las que se debe elegir una.
- Los puntos suspensivos … indican repetición varias veces de la opción anterior.

En SQL no se distingue entre mayúsculas y minúsculas. Da lo mismo como se escriba.

El final de una instrucción o sentencia lo marca el signo de punto y coma (;).

Las sentencias SQL (SELECT, INSERT, …) se pueden escribir en varias líneas siempre que las palabras no sean partidas.

Los comentarios en el código SQL pueden ser de 2 tipos:

- de bloque: comienzan por /* y terminan por */
- de línea: comienzan por – y terminan en final de línea

Ejemplos:

```
/*
  Esto es un comentario
  de varias líneas.

  Fin.
*/

-- Esto es un comentario de una línea
```

3. Manipulación de Bases de Datos

3.1. Crear una base de datos

En MySQL, para crear una base de datos se usa el comando CREATE DATABASE:

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nombre_base_datos
    [DEFAULT] CHARACTER SET juego_caracteres |
    [DEFAULT] COLLATE nombre_colación
    ;
```

- `DATABASE` y `SCHEMA` son sinónimos.
- `IF NOT EXISTS` crea la base de datos sólo si no existe una base de datos con el mismo nombre.
- `CHARACTER SET` : Especifica el set de caracteres que vamos a utilizar en la base de datos.
- `COLLATE` : Especifica el tipo de cotejamiento que vamos a utilizar en la base de datos. Indica el criterio que vamos a seguir para ordenar las cadenas de caracteres.

Ejemplos:

Si no especificamos el set de caracteres en la creación de la base de datos, se usará `latin1` por defecto.

```
CREATE DATABASE nombre_base_datos;
```

Las bases de datos que vamos a crear durante el curso usarán el set de caracteres `utf8` o `utf8mb4` .

```
CREATE DATABASE nombre_base_datos CHARACTER SET utf8;
```

El cotejamiento, es el criterio que vamos a utilizar para ordenar las cadenas de caracteres de la base de datos. Si no especificamos ninguno se usará el que tenga asignado por defecto el set de caracteres escogido. Por ejemplo, para el set de caracteres `utf8` se usa `utf8_general_ci` .

El siguiente ejemplo muestra cómo podemos especificar el cotejamiento que queremos de forma explícita:

```
CREATE DATABASE nombre_base_datos CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Para ver cuáles son los sets de caracteres que tenemos disponibles podemos usar la siguiente sentencia:

```
SHOW CHARACTER SET;
```

Para consultar qué tipos de cotejamiento hay disponibles podemos usar:

```
SHOW COLLATION;
```

El cotejamiento puede ser:

- case-sensitive (_cs): Los caracteres `a` y `A` son diferentes.
- case-insensitive (_ci): Los caracteres `a` y `A` son iguales.
- binary (_bin): Dos caracteres son iguales si los valores de su representación numérica son iguales.

3.1.1. Conceptos básicos sobre la codificación de caracteres

Unicode es un set de caracteres universal, un estándar en el que se definen todos los caracteres necesarios para la escritura de la mayoría de los idiomas hablados en la actualidad. El estándar Unicode describe las propiedades y algoritmos necesarios para trabajar con los caracteres Unicode y este estándar es gestionado por el consorcio Unicode.

Los formatos de codificación que se pueden usar con Unicode se denominan **UTF-8**, **UTF-16** y **UTF-32**.

- **UTF-8** utiliza 1 byte para representar caracteres en el set ASCII, 2 bytes para caracteres en otros bloques alfabéticos y 3 bytes para el resto del BMP (Basic Multilingual Plane), que incluye la mayoría de los caracteres utilizados frecuentemente. Para los caracteres complementarios se utilizan 4 bytes.
- **UTF-16** utiliza 2 bytes para cualquier carácter en el BMP y 4 bytes para los caracteres complementarios.
- **UTF-32** emplea 4 bytes para todos los caracteres.

Se recomienda la lectura del artículo Codificación de caracteres: conceptos básicos publicado por la W3C.org.

En MySQL el set de caracteres `utf8` utiliza **un máximo de 3 bytes por carácter** y contiene sólo los caracteres del BMP (Basic Multilingual Plane). Según el estándar Unicode, el formato de codificación `utf8` permite representar caracteres desde 1 hasta 4 bytes, esto quiere decir que **el set de caracteres `utf8` de MySQL no permite almacenar caracteres Unicode con 4 bytes**.

Este problema se solucionó a partir de MySQL 5.5.3, cuando se añadió el set de caracteres `utf8mb4` que permite utilizar hasta 4 bytes por carácter.

Por ejemplo, en MySQL los caracteres Emoji Unicode no se podrían representar con `utf8`, habría que utilizar `utf8mb4`:

Emoji	Unicode	Bytes (UTF-8)
😃	U+1F603	`\xF0\x9F\x98\x83`
🙈	U+1F648	`\xF0\x9F\x99\x88`
👾	U+1F47E	`\xF0\x9F\x91\xBE`

3.2. Eliminar una base de datos

```
DROP {DATABASE | SCHEMA} [IF EXISTS] nombre_base_datos;
```

- `DATABASE` y `SCHEMA` son sinónimos.
- `IF EXISTS` elimina la la base de datos sólo si ya existe.

Ejemplo:

```
DROP DATABASE nombre_base_datos;
```

3.3. Modificar una base de datos

```
ALTER {DATABASE | SCHEMA} [nombre_base_datos]
  alter_specification [, alter_especification] ...
```

Solo permite modificar el juego de caracteres y su colación (`CHARACTER SET` y `COLLATE`)

Ejemplo:

```
ALTER DATABASE nombre_base_datos CHARACTER SET utf8;
```

3.4. Consultar el listado de bases de datos disponibles

```
SHOW DATABASES;
```

Muestra un listado con todas las bases de datos a las que tiene acceso el usuario con el que hemos conectado a MySQL.

3.5. Seleccionar una base de datos

```
USE nombre_base_datos;
```

Se utiliza para indicar la base de datos con la que queremos trabajar.

3.6. Mostrar la sentencia SQL de creación de una base de datos

```
SHOW CREATE DATABASE nombre_base_datos;
```

Se puede utilizar para visualizar la sentencia SQL que sería necesaria ejecutar para crear la base de datos que le estamos indicando como parámetro.

4. Manipulación de tablas

La base de datos creada hasta el momento, no es más que una estructura vacía. El siguiente paso será crear las tablas que contendrán los datos.

En este apartado veremos los comandos SQL que se utilizarán para crear y modificar la definición de una tabla, así como para eliminarla de la base de datos.

4.1. Tipos de datos

Antes de empezar con las sentencias para la creación de tablas, es necesario conocer los tipos de datos disponibles en MySQL para poder identificar qué tipo de dato es el más adecuado para cada atributo de las entidades.

Los tipos de datos en MySQL se pueden clasificar en tres grandes grupos:

a) Tipo Numérico (enteros, en coma flotante y en coma flotante)

- `TINYINT` : Tipo entero muy pequeña
- `SMALLINT` : Tipo entero pequeña
- `MEDIUMINT` : Tipo entero mediana
- `INT` : Tipo entero (en MySQL `INT` es equivalente a `INTEGER`)
- `BIGINT` : Tipo Entero largo
- `FLOAT` : Número único de precisión de coma flotante de 4 bytes, almacena valores aproximados.
- `DOUBLE` : Número doble de precisión de coma flotante de 8 bytes, almacena valores aproximados.
- `DECIMAL (M,D)` : Número de coma fija, almacena valores exactos (sin redondeos).

En MySQL los tipos de datos `DECIMAL` y `NUMERIC` son equivalentes.

- `M` indica el número de dígitos en total (la precisión). Tiene un rango de 1 a 65.
- `D` es el número de cifras decimales. Tiene un rango de 0 a 30.

b) Tipo Cadena

- `CHAR (n)` : Cadena con caracteres con longitud fija comprendida entre 0 y 255.
- `VARCHAR(n)` : Cadena con caracteres con longitud variable comprendida entre 0 y 65535.
- `TINYBLOB` : BLOB (1) muy pequeño
- `TEXT[(n)]` : Cadena de texto con longitud máxima de 65536.
- `MEDIUMTEXT` : Cadena de texto mediana con longitud máxima de 16 MB.
- `LONGTEXT` : Cadena de texto larga con longitud máxima de 4 GB.
- `ENUM (valor1, valor2, ...)` : Enumeración de valores, pueden tener 65535 valores, pero sólo permite seleccionar un valor de la lista.
- `SET (valor1, valor2, ...)` : Conjunto de valores, pueden tener 64 valores, pero permite seleccionar varios valores de la lista.

c) Tipo Fecha

- `DATE` : Fecha con formato AAAA-MM-DD
- `TIME` : Hora con formato hh:mm:ss
- `DATETIME` : Fecha y hora con formato AAAA-MM-DD hh:mm:ss
- `TIMESTAMP` : Lapso de tiempo con formato AAAAMMDDhhmmss
- `YEAR [2|4]` : Año con formato AA |AAAA (por defecto se almacena con 4 dígitos)

Puedes encontrar más información sobre tipos de datos en la documentación oficial de MySQL.

4.2. Crear una tabla

El nombre de las tablas debe cumplir las siguientes reglas:

- Deben comenzar con una letra.
- No deben tener más de 30 caracteres.
- Sólo se permiten utilizar letras del alfabeto (inglés), números o el signo de subrayado (*también el signo $ y #, pero esos se utilizan de manera especial por lo que no son recomendados*)

- No puede haber dos tablas con el mismo nombre.
- No puede coincidir con el nombre de una palabra reservada de SQL.

Para la creación de tablas con SQL se utiliza el comando `CREATE TABLE`. Este comando tiene una sintaxis más compleja de la que aquí se expone, pero vamos a comenzar por **versión simplificada** de la sintaxis necesaria para la creación de una tabla en MySQL. Para una definición más exhaustiva, puede consultar la sintaxis de creación de tablas en la documentación oficial de MySQL.

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] nombre_tabla (
   nombre_columna1 tipo_de_dato  [restricciones de columna1],
   nombre_columna2 tipo_de_dato  [restricciones de columna2],
   nombre_columna3 tipo_de_dato  [restricciones de columna3],
   ...
   [restricciones de tabla]
 );
```

Para realizar las separaciones se utiliza la coma. **La última línea, antes del paréntesis de cierre, no lleva coma**.

Lo veremos mejor con un ejemplo:

```
DROP DATABASE IF EXISTS proveedores;
CREATE DATABASE proveedores CHARSET utf8mb4;
USE proveedores;

CREATE TABLE categoria (
  codigo INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL
);

CREATE TABLE pieza (
  codigo INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  color VARCHAR(50) NOT NULL,
  precio DECIMAL(7,2) NOT NULL CHECK (precio > 0),
  codigo_categoria INT UNSIGNED NOT NULL,
  FOREIGN KEY (codigo_categoria) REFERENCES categoria(codigo)
);
```

4.2.1. Restricciones sobre las columnas de la tabla

Podemos aplicar las siguientes restricciones sobre las columnas de la tabla:

- `PRIMARY KEY`: Para indicar que una columna o conjunto de columnas cuyo valor identifica unívocamente a cada fila. Debe ser única, no nula y es obligatoria. Como máximo podremos definir una clave primaria por tabla y es muy recomendable definirla. Lo normal es definirlas como en el ejemplo, pero podemos definirlas al final de la definición de las columnas. Hay que tener en cuenta que a la hora de definir claves primarias compuestas (la componen 2 ó más campos de la tabla), ésta deberá ser definida forzosamente tras la definición de los campos involucrados, siguiendo esta sintaxis:

  ```
  PRIMARY KEY (id_turista,id_hotel)
  ```

- `NOT NULL` o `NULL`: Indica si la columna permite almacenar valores nulos o no.
- `DEFAULT`: Permite indicar un valor inicial por defecto si no especificamos ninguno en la inserción.

```
fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
nombre VARCHAR(250) DEFAULT 'Sin nombre',
```

- `AUTO_INCREMENT`: Sirve para indicar que es una columna autonumérica. Su valor se incrementa automáticamente en cada inserción de una fila. Sólo se utiliza en campos de tipo entero.
- `UNIQUE KEY`: Indica que el valor de la columna es único y no pueden aparecer dos valores iguales en la misma columna.
- `CHECK`: Nos permite realizar restricciones sobre una columna. En las versiones previas a MySQL 8.0 estas restricciones no se aplicaban. A partir de la versión de MySQL 8.0 ya sí se aplican las restricciones definidas con `CHECK`.

```
precio FLOAT CHECK (edad > 0),
```

- `FOREIGN KEY`: Nos permite definir una clave ajena de la tabla respecto de otra tabla. No se puede definir una **restricción de integridad referencial** que se refiere a una tabla antes de que dicha tabla haya sido creada. Es importante resaltar que una clave ajena debe referenciar a una clave primaria completa de la tabla padre, y nunca a un subconjunto de las columnas que forman esta clave primaria. Además, el valor de la/s columna/s será NULL o bien el valor de la clave primara de la tabla a la que hace referencia.
- `REFERENCES`: Se utiliza en la definición de la **restricción de integridad referencial** (`FOREIGN KEY`) para indicar la tabla a la que hace referencia.

En la definición de una tabla pueden aparecer varias cláusulas `FOREIGN KEY`, tantas como **claves ajenas** tenga la tabla, sin embargo sólo puede existir una **clave primaria** (`PRIMARY KEY`). Como ocurre con las claves primarias, si las claves ajenas son compuestas, se definen forzosamente al final de las definiciones de las columnas de la tabla, de la siguiente forma:

```
FOREIGN KEY (id_curso, id_aula) REFERENCES cursos (id, num_aula)
```

4.2.2. Opciones en la declaración de claves ajenas (`FOREING KEY`)

A la hora de definir las claves ajenas, hay que establecer que acciones se llevarán a cabo en el caso de que se eliminen (`ON DELETE`) o se modifiquen (`ON UPDATE`) las claves primarias referencias.

`ON DELETE` y `ON UPDATE`: Nos permiten indicar el efecto que provoca el borrado o la actualización de los datos que están referenciados por claves ajenas. Las opciones que podemos especificar son las siguientes:

- `NO ACTION`: Es una palabra clave del estándar SQL. En MySQL es equivalente a `RESTRICT`.
- `RESTRICT`: Impide que se puedan actualizar o eliminar las filas que tienen valores referenciados por claves ajenas. Es la **opción por defecto en MySQL**.
- `CASCADE`: Permite actualizar o eliminar las filas que tienen valores referenciados por claves ajenas.
- `SET NULL`: Asigna el valor `NULL` a las filas que tienen valores referenciados por claves ajenas.
- `SET DEFAULT`: No es posible utilizar esta opción cuando trabajamos con el motor de almacenamiento **InnoDB**.

Puedes encontrar más información en la documentación oficial de MySQL.

Aplicadas a nuestro ejemplo:

```
DROP DATABASE IF EXISTS proveedores;
CREATE DATABASE proveedores CHARSET utf8mb4;
USE proveedores;

CREATE TABLE categoria (
  codigo INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL
);

CREATE TABLE pieza (
  codigo INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  color VARCHAR(50) NOT NULL,
  precio DECIMAL(7,2) NOT NULL,
  codigo_categoria INT UNSIGNED NOT NULL,
  FOREIGN KEY (codigo_categoria) REFERENCES categoria(codigo)
  ON DELETE CASCADE ON UPDATE CASCADE
);
```

La acción a realizar sobre `ON DELETE` no tiene porque ser la misma que sobre `ON UPDATE` .

4.2.3. Criterios de notación para los nombres de restricciones

La utilización de la cláusula **CONSTRAINT nombre_restricción** establece un nombre determinado para cada restricción, lo cual permite buscar en el Diccionario de Datos de la base de datos con posterioridad y fácilmente las restricciones introducidas para una determinada tabla. Sino se le asigna un nombre, el SGBD le asignará uno aleatorio.

Se puede utilizar para todo tipo de restricciones, siendo más habitual hacerlo en las restricciones de clave ajena o las definidas por el usuario (haciendo uso de la cláusula `CHECK`).

Los criterios o normas que se suelen seguir para dar nombre a estas restricciones son:

Clave Primaria:

```
  CONSTRAINT tabla_campo_pk PRIMARY KEY (campo)
```

Ejemplo:

```
  CONSTRAINT pieza_codigo_pk PRIMARY KEY (codigo)
```

Clave Ajena:

```
CONSTRAINT tabla1_tabla2_fk1 FOREIGN KEY (campo) REFERENCES tabla2 (clave_primaria)
```

Ejemplo:

```
CONSTRAINT pieza_categoria_fk1 FOREIGN KEY (cod_cat) REFERENCES CATEGORIA (codigo)
```

Restricciones definidas por el usuario:

```
CONSTRAINT tabla_campo_ck1 CHECK (condiciones)
```

Ejemplo:

```
CONSTRAINT pieza_color_ck1 CHECK (color IN ('Rojo', 'Azul', 'Amarillo', 'Blanco',
'Negro'))
```

4.2.4. Opciones a tener en cuenta en la creación de las tablas

Algunas de las opciones que podemos indicar durante la creación de las tablas son las siguientes:

- `AUTO_INCREMENT` : Aquí podemos indicar el valor inicial que vamos a usar en el campo definido como `AUTO_INCREMENT` .
- `CHARACTER SET` : Especifica el set de caracteres que vamos a utilizar en la tabla.
- `COLLATE` : Especifica el tipo de cotejamiento que vamos a utilizar en la tabla.
- `ENGINE` : Especifica el motor de almacenamiento que vamos a utilizar para la tabla. Los más habituales en MySQL son **InnoDB** y **MyISAM**. Por defecto las tablas se crean con el motor **InnoDB**

Para conocer todas las opciones posibles podemos consultar la sintaxis de creación de tablas en la documentación oficial de MySQL. Con el objetivo de simplificar la creación de tablas solamente hemos enumerado las opciones con las que vamos a trabajar durante el curso.

En la documentación oficial de MySQL podemos encontrar más información sobre los diferentes motores de almacenamiento disponibles en MySQL.

Ejemplo:

```
DROP DATABASE IF EXISTS proveedores;
CREATE DATABASE proveedores CHARSET utf8mb4;
USE proveedores;

CREATE TABLE categoria (
  codigo INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL
)  ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1000;

CREATE TABLE pieza (
  codigo INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  color VARCHAR(50) NOT NULL,
  precio FLOAT NOT NULL,
  codigo_categoria INT UNSIGNED NOT NULL,
  FOREIGN KEY (codigo_categoria) REFERENCES categoria(codigo)
)  ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1000;
```

En este ejemplo se ha seleccionado para cada una de las tablas las siguientes opciones de configuración: **InnoDB** como motor de base de datos, **utf8** como el set de caracteres y el valor **1000** como valor inical para las columnas de tipo **AUTO_INCREMENT**.

4.3. Eliminar una tabla

La sintaxis de la sentencia SQL para eliminar tablas es:

```
DROP [TEMPORARY] TABLE [IF EXISTS] nombre_tabla [, nombre_tabla];
```

Ejemplos:

```
DROP TABLE nombre_tabla;
DROP TABLE IF EXISTS nombre_tabla;
DROP TABLE nombre_tabla_1, nombre_tabla_2;
```

Peligro

El borrado de una tabla es irreversible y no hay ninguna petición de confirmación, por lo que conviene ser muy cuidadoso con esta operación. Al borrar una tabla se borran todos los datos que contiene.

4.4. Modificar una tabla

En muchas ocasiones es necesario modificar los atributos de una tabla, añadir nuevos campos o eliminar otros. Si la tabla no tiene datos podemos eliminar la tabla y volver a crearla, pero si se trata de una tabla que ya contiene datos tenemos que hacer uso de la sentencia `ALTER TABLE`.

A continuación se muestra la sintaxis básica para la modificación de una tabla en MySQL. Puede consultar la sintaxis de modificación de tablas en la documentación oficial de MySQL.

```
ALTER TABLE <nombre_tabla>
       [ MODIFY [COLUMN] col_name column_definition]
       [ CHANGE <nombre_columna> <definicion_columna> ]
       [ ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT} ]
       [ DROP COLUMN <nombre_columna> ]
       [ ADD [COLUMN] <definicion_columna> ]
       [ ADD CONSTRAINT <restriccion> ]
```

4.4.1. MODIFY

Nos permite modificar el tipo de dato de una columna y sus atributos.

Suponemos que tenemos la siguiente tabla creada

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(25)
);
```

Y queremos modificar las columna `nombre` para que pueda almacenar 50 caracteres y además que sea `NOT NULL`. En este caso usaríamos la sentencia:

```
ALTER TABLE usuario MODIFY nombre VARCHAR(50) NOT NULL;
```

Después de ejecutar esta sentencia la tabla quedaría así:

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL
);
```

4.4.2. CHANGE

Nos permite renombrar una columna, modificar el tipo de dato de una columna y sus atributos.

Suponemos que tenemos la siguiente tabla creada

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre_de_usuario VARCHAR(25)
);
```

Y queremos renombrar el nombre de la columna `nombre_de_usuario` como `nombre`, que pueda almacenar 50 caracteres y además que sea `NOT NULL`. En este caso usaríamos la sentencia:

```
ALTER TABLE usuario CHANGE nombre_de_usuario nombre VARCHAR(50) NOT NULL;
```

Después de ejecutar esta sentencia la tabla quedaría así:

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL
);
```

4.4.3. ALTER

nos permite asignar un valor por defecto a una columna o eliminar el valor por defecto que tenga establecido.

Suponemos que tenemos la siguiente tabla creada

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  sexo ENUM('H', 'M') NOT NULL
);
```

Y queremos que el valor por defecto de la columna `sexo` sea `M`. En este caso usaríamos la sentencia:

```
ALTER TABLE usuario ALTER sexo SET DEFAULT 'M';
```

Después de ejecutar esta sentencia la tabla quedaría así:

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  sexo ENUM('H', 'M') NOT NULL DEFAULT 'M'
);
```

Si ahora quisiéramos eliminar el valor por defecto de la columna `sexo`, usaríamos la siguiente sentencia:

```
ALTER TABLE usuario ALTER sexo DROP DEFAULT;
```

4.4.4. DROP

Nos permite eliminar una columna de la tabla.

Suponemos que tenemos la siguiente tabla creada

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  apellido1 VARCHAR(50) NOT NULL,
  apellido2 VARCHAR(50),
  sexo ENUM('H', 'M') NOT NULL DEFAULT 'M',
  fecha_nacimiento DATE NOT NULL
);
```

Y queremos eliminar la columna `fecha_nacimiento`. En este caso usaríamos la sentencia:

```
ALTER TABLE usuario DROP COLUMN fecha_nacimiento;
```

Después de ejecutar esta sentencia la tabla quedaría así:

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  apellido1 VARCHAR(50) NOT NULL,
  apellido2 VARCHAR(50),
  sexo ENUM('H', 'M') NOT NULL DEFAULT 'M'
);
```

4.4.5. ADD

Nos permite añadir nuevas columnas a una tabla. Con los modificadores `FIRST` y `AFTER` podemos elegir el lugar de la tabla donde queremos insertar la nueva columna. `FIRST` coloca la nueva columna en primer lugar y `AFTER` la colocaría detrás de la columna que se especifique. Si no se especifica nada la nueva columna se añadiría detrás de la última columna de la tabla.

Suponemos que tenemos la siguiente tabla creada

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  sexo ENUM('H', 'M') NOT NULL
);
```

Y queremos añadir la columna `fecha_nacimiento` de tipo `DATE`:

```
ALTER TABLE usuario ADD COLUMN fecha_nacimiento DATE NOT NULL;
```

En este caso la nueva columna se ha añadido detrás de la última columna, `sexo`. La tabla quedaría así:

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  sexo ENUM('H', 'M') NOT NULL,
  fecha_nacimiento DATE NOT NULL
);
```

Suponemos que ahora queremos añadir las columnas `apellido1` y `apellido2` detrás de la columna `nombre`.

```
ALTER TABLE usuario ADD COLUMN apellido1 VARCHAR(50) NOT NULL AFTER nombre;
ALTER TABLE usuario ADD COLUMN apellido2 VARCHAR(50) AFTER apellido1;
```

Después de ejecutar todas las sentencias la tabla quedaría así:

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  apellido1 VARCHAR(50) NOT NULL,
  apellido2 VARCHAR(50),
  sexo ENUM('H', 'M') NOT NULL DEFAULT 'M',
  fecha_nacimiento DATE NOT NULL
);
```

4.4.6. Añadir, Modificar y Eliminar Restricciones

Sabemos que una restricción es una condición de obligado cumplimiento para una o más columnas de la tabla. A cada restricción se le pone un nombre, en el caso de no poner un nombre (en las que eso sea posible) entonces el propio SGBD le coloca el nombre que es un nemotécnico con el nombre de tabla, columna y tipo de restricción.

Hemos visto que se pueden añadir al crear la tabla, o bien, podemos hacerlo mediante modificación posterior de la tabla. También se puede modificar una restricción creada.

Su sintaxis general es:

```
ALTER TABLE nombre_tabla { ADD | MODIFY | DROP}
  CONSTRAINT nombre_restricción1   tipo_restricción  (columnas);
```

Suponemos que tenemos la siguiente tabla creada:

```
CREATE TABLE usuario (
  DNI VARCHAR(9) NOT NULL,
  nombre VARCHAR(50) NOT NULL,
  apellido1 VARCHAR(50) NOT NULL,
  apellido2 VARCHAR(50),
  sexo ENUM('H', 'M') NOT NULL DEFAULT 'M',
  fecha_nacimiento DATE NOT NULL
);
```

Para añadir la restricción de clave primaria utilizaríamos la siguiente sentencia:

```
ALTER TABLE usuario ADD CONSTRAINT ususario_pk PRIMARY KEY (DNI);
```

Para definir una clave ajena a otra tabla, primero necesitaríamos añadir la columna correspondiente, si no está en nuestra tabla, y a continuación añadir la restricción. Las sentencias serían la siguientes:

```
ALTER TABLE usuario ADD COLUMN cod_tabla INT;
ALTER TABLE usuario ADD CONSTRAINT user_tabla_fk FOREIGN KEY (cod_tabla) REFERENCES
tabla (cod);
```

4.5. Otras operaciones con tablas

Además de las operaciones habituales anteriores, existen otras acciones que pueden que nos sean de gran utilidad:

a) Para consultar el **listado de tablas** disponibles en la base de datos con la que estamos trabajando usaremos la sentencia:

```
SHOW TABLES;
```

b) Para mostrar información sobre la **estructura de una tabla** podemos utilizar alguna de las siguientes sentencias:

```
DESCRIBE nombre_tabla;
```

```
DESC nombre_tabla;
```

c) Para visualizar la **sentencia SQL** que sería necesaria ejecutar para **crear la tabla** que le estamos indicando como parámetro, podemos utilizar la siguiente sentencia:

```
SHOW CREATE TABLE nombre_tabla;
```

5. Usuarios y gestión de privilegios (DCL).

Una de las funciones del SGBD es la de proporcionar seguridad en el acceso a los datos a través de mecanismos de control de acceso.

En SQL, y así lo hacen todos los SGBD relacionales, se sigue un modelo Usuario-Privilegio para otorgar acceso a los objetos de la Base de Datos. Existen una serie de privilegios predefinidos y es el administrador del SGBD el encargado de asignar o no los privilegios a los usuarios sobre determinados objetos (tablas, procedimientos, . . .).

Supongamos que somos los administradores de un SGBD MySQL y tenemos que proporcionar acceso a una Base de Datos para una aplicación *empresa* a un desarrollador de mi compañía:

```
-- Si no hemos creado la base de datos, podemos hacerlo ahora
CREATE DATABASE proveedores;
-- Crea el usuario asignándole contraseña
CREATE USER 'desarrollador' IDENTIFIED BY 'micontraseña';
-- Asigna todos los privilegios al usuario sobre la base de datos
GRANT ALL PRIVILEGES ON proveedores.* TO desarrollador;
```

Así, hemos creado la Base de Datos y el usuario, y hemos concedido todos los privilegios a dicho usuario sobre esa Base de Datos.

Utilizaremos el **DCL** (*Data Contol Language*) o **Lenguaje de Control de datos**, para reforzar la seguridad de la base de datos en un entorno de múltiples usuarios.

Los comandos DCL que podemos utilizar para **gestionar los permisos de usuario** son `GRANT` y `REVOKE`

`GRANT` Permite conceder privilegios sobre un objeto a un usuario de la Base de Datos.

```
GRANT <privilegio> ON <objeto> TO <usuario>
[WITH GRANT OPTIONS]
```

`REVOKE` Permite eliminar el privilegio sobre un objeto a un usuario.

```
REVOKE <privilegio> ON <objeto> FROM <usuario>
```

Los privilegios que se pueden otorgar y/o revocar son:

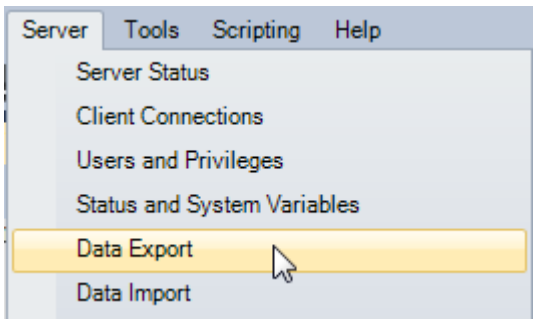
- Privilegios sobre objetos: `SELECT`, `INSERT`, `DELETE` o `UPDATE`.
- Privilegios del sistema: `CREATE`, `DROP` y `ALTER`.
- La opción `WHIT GRANT OPTIONS`, indica que se le otorga a dicho usuario la posibilidad de otorgar y revocar permisos a otros usuarios

6. Exportar e Importar Bases de Datos

A continuación explicaremos como exportar e importar el script de nuestra base de datos.

6.1. Exportar Base de datos

Para hacer una copia de respaldo o también llamado Backup de una de las base de datos (ó de todas), debemos ir al menú superior y seleccionar **Server->Data Export**:



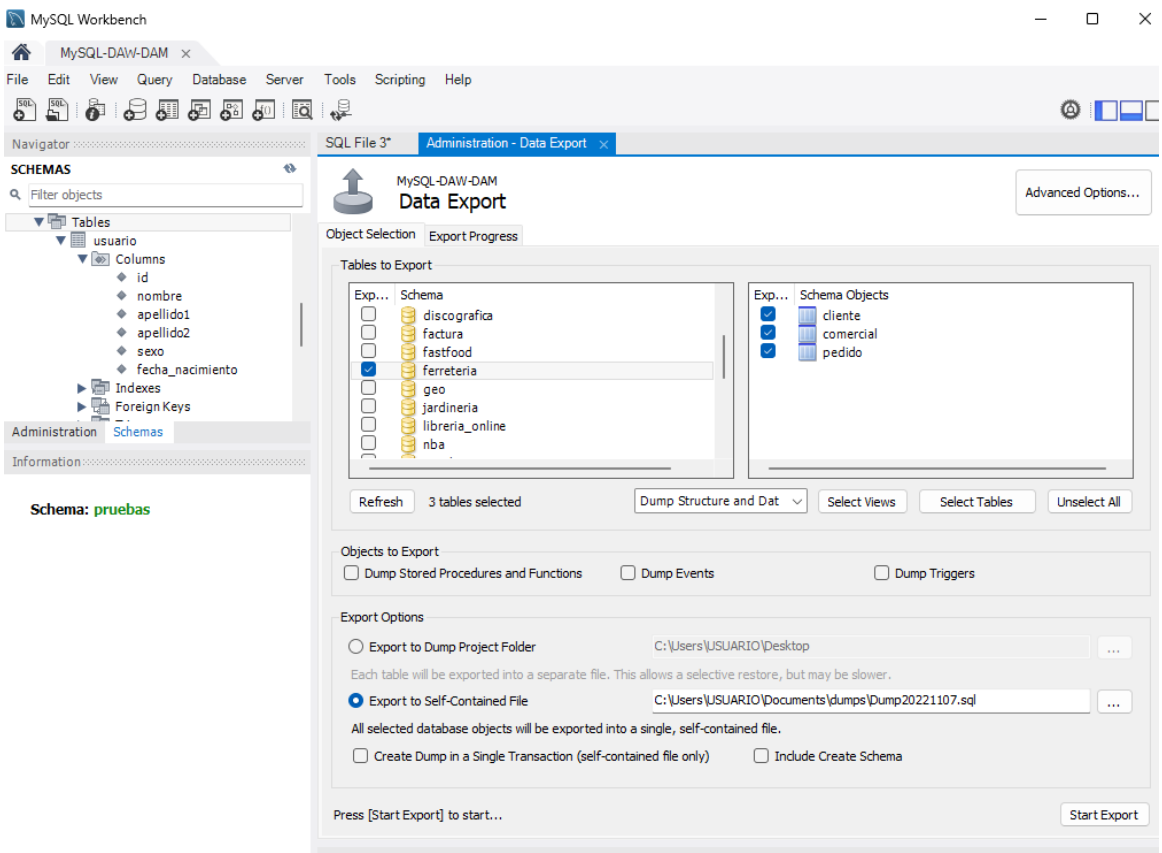
Seleccionamos la base de datos y activaremos la opción de **"Dump Structure and Data**". Esta opción activa la *copia de las tablas y de los datos de las tablas*, es decir **TODO** el contenido de la base de datos.

Tenemos dos opciones para exportar:

a) **Export to Dump Project Folder**: esta opción nos crea un archivo por cada tabla de nuestra base de datos y los guarda en una carpeta.

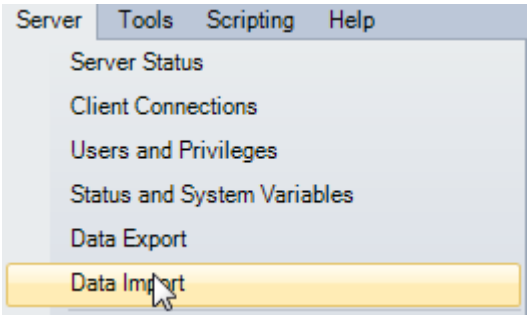
b) **Export to Self-Contained File**: nos crea un único fichero SQL.

Para simplificar la salida, exportaremos a un único fichero SQL. Para ello seleccionaremos la opción de "**Export to Self-Contained File**" y pulsaremos sobre "**Star Export**":



6.2. Importar Base de Datos

Para importar ó restaurar una base de datos debemos ir al menú superior y hacer click en **Server->Data Import**:



Como el fichero que hemos generado de backup es un único fichero, seleccionaremos la opción de "**Import from Self-Contained File**". Como la base de datos no existe, debemos crearla en el botón de "**New...**" Le daremos un nombre y pulsaremos "**Start Import**":