# UT5.Modelo Físico: SQL - Instrucciones DDL y DML



## 1. INTRODUCCIÓN

En esta unidad se describe la metodología de diseño físico para bases de datos relacionales. En esta etapa, se parte del esquema lógico global obtenido durante el diseño lógico y se obtiene una descripción de la implementación de la base de datos en memoria secundaria. Esta descripción es completamente dependiente del SGBD específico que se vaya a utilizar.

El diseño de una base de datos se descompone en tres etapas: diseño conceptual, lógico y físico. La etapa del diseño lógico es independiente de los detalles de implementación y dependiente del tipo de SGBD que se vaya a utilizar. La salida de esta etapa es el esquema lógico global y la documentación que lo describe. Todo ello es la entrada para la etapa que viene a continuación, el diseño físico.

Mientras que en el diseño lógico se especifica qué se guarda, en el diseño físico se especifica cómo se guarda. Para ello, el diseñador debe conocer muy bien toda la funcionalidad del SGBD concreto que se vaya a utilizar y también el sistema informático sobre el que éste va a trabajar.

## 2. LENGUAJE SQL

A nivel teórico, existen dos lenguajes para el manejo de bases de datos:

- DDL (Data Definition Language) Lenguaje de definición de datos. Es el lenguaje que se usa para crear bases de datos y tablas, y para modificar sus estructuras, así como los permisos y privilegios.

  Este lenguaje trabaja sobre unas tablas especiales llamadas diccionario de datos.

- DML (Data Manipilation Language) lenguaje de manipulación de datos. Es el que se usa para modificar y obtener datos desde las bases de datos.

SQL engloba ambos lenguajes DDL+DML, y los estudiaremos juntos, ya que ambos forman parte del conjunto de sentencias de SQL.

### 2.1. LENGUAJE DDL

Prácticamente, la creación de la base de datos consiste en la creación de las tablas que la componen. En realidad, antes de poder proceder a la creación de las tablas, normalmente hay que crear la base de datos, lo que a menudo significa definir un espacio de nombres separado para cada conjunto de tablas. De esta manera, para una SGBD *(Sistema Gestor de Base de Datos)* se pueden gestionar diferentes bases de datos independientes al mismo tiempo sin que se den conflictos con los nombres que se usan en cada una de ellas.

Cada conjunto de relaciones que componen un modelo completo forma una base de datos. Desde el punto de vista de SQL, una base de datos es sólo un conjunto de relaciones (o tablas), y para organizarlas o distinguirlas se accede a ellas mediante su nombre. A nivel de sistema operativo, cada base de datos se guarda en un directorio diferente.

Debido a esto, crear una base de datos es una tarea muy simple. Claro que, en el momento de crearla, la base de datos estará vacía, es decir, no contendrá ninguna tabla.

Vamos a crear y manipular nuestra propia base de datos, al tiempo que nos familiarizamos con la forma de trabajar de MySQL o MariaDB.

## 2.1.1. CREAR BASES DE DATOS

Para empezar, crearemos una base de datos para nosotros solos, y la llamaremos "prueba". Para crear una base de datos se usa una sentencia **CREATE DATABASE**:

```
CREATE DATABASE [IF NOT EXISTS] nombre_db
        [CHARACTER SET juego_caracteres]
            [COLLATE nombre_colación];
```

Las opciones entre corchetes indican opcionalidad *(por tanto, el usuario no está obligado a indicarlas)*.

Con la opción **CHARACTER SET** indicamos el juego de caracteres que va a usar la base de datos *(utf8, latin1, latin2, etc.)* y **COLLATE** indica el tipo de ordenación, es decir, especifica cómo va a tratar el alfabeto del juego de caracteres, cómo se ordena (por ejemplo, la ñ después de la n y no conforme la tabla de códigos ascii).

```
mysql> CREATE DATABASE prueba
            CHARACTER SET utf8
            COLLATE utf8_spanish_ci;
Query OK, 1 row affected (0.03 sec)

mysql>
```

Podemos averiguar cuántas bases de datos existen en nuestro sistema usando la sentencia **SHOW DATABASES**:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| mysql              |
| prueba             |
| test               |
+--------------------+
3 rows in set (0.00 sec)

mysql>
```

Para seleccionar una base de datos se usa el comando **USE**, que no es exactamente una sentencia SQL, sino más bien de una opción de **MariaDB**.

```
mysql> USE prueba;
Database changed
mysql>
```

## 2.1.2. BORRADO DE BASES DE DATOS

El comando **DROP DATABASE** es el utilizado para eliminar bases de datos. En MariaDB, el comando se acompaña del nombre de la base de datos a eliminar, por ejemplo:

```
mysql> DROP DATABASE prueba;
```

## 2.1.3. CREACIÓN DE TABLAS

Veamos ahora la sentencia CREATE TABLE que sirve para crear tablas.

La sintaxis de esta sentencia es muy compleja, ya que existen muchas opciones y tenemos muchas posibilidades diferentes a la hora de crear una tabla.

```
CREATE TABLE nombre_tabla
        [(definición_create, ...)]
```

La porción **definición_create** especificará la definición de los campos que va a contener la tabla y sus restricciones. Los puntos suspensivos sugieren que **definición_create** se puede repetir tantas veces como sea necesario.

```
definición_create:
        definición_columna
| [CONSTRAINT nombre_restricción] PRIMARY KEY (nombre_columna, ...)
| [CONSTRAINT nombre_restricción] FOREIGN KEY (nombre_columna, ...)
        [definicion_referencia]
```

La primera cláusula que lleva la definición es **definición_columna**, donde especificará la definición de un campo o columna de la tabla.

```
definición columna:
        nombre_columna tipo_datos [NOT NULL | NULL] [DEFAULT valor]
[AUTO_INCREMENT]
```

NOT NULL y NULL especifican que la columna admite o no admite valores nulos, es decir si un campo puede quedar sin valor, o con valor desconocido.

DEFAULT indica el valor por defecto que toma el campo si no es especificado de forma explícita, por ejemplo,

```
codigo_postal  INT(5) NOT NULL DEFAULT 13600
```

En cuanto a la parte de **definición_referencia** sirve para crear una clave foránea. De esta manera se enlaza un campo a su campo origen, es decir, se crea una referencia.

```
definición_referencia:
        REFERENCES nombre_tabla (nombre_columna, ...)
                [ON DELETE {CASCADE | SET NULL | NO ACTION |
RESTRICT} ]
                [ON UPDATE {CASCADE | SET NULL | NO ACTION |
RESTRICT} ]
```

Las opciones ON DELETE y ON UPDATE establecen el comportamiento del gestor en caso de que las filas de la tabla padre (es decir, la tabla referenciada) se borren o se actualicen. Los comportamientos puedes ser CASCADE, SET NULL, NO ACTION y RESTRICT.

- Si se usa NO ACTION o RESTRICT y se intenta un borrado o actualización sobre la tabla padre, la operación se impide, rechazando el borrado o la actualización.
- Si se especifica CASCADE, la operación se propaga en cascada a la tabla hija, es decir, si se actualiza la tabla padre, se actualizan los registros relacionados de la tabla hija, y si se borra un registro de la tabla padre, se borran aquellos registros de la tabla hija que estén referenciando al registro borrado.
- Si se indica SET NULL, se establece a NULL la clave foránea afectada por un borrado o una modificación de la tabla padre.

Si no se especifica ON DELTE y ON UPDATE por defecto se actúa como **NO ACTION**.

Ejemplo de creación de una tabla:

```
mysql> CREATE TABLE IF NOT EXISTS alumnos(
        dni VARCHAR(9) NOT NULL,
        nombre VARCHAR(20) NOT NULL DEFAULT 'alumno genérico'
        apellidos VARCHAR(50),
        cp INT(5) DEFAULT 13600,
        codCurso INT(3),
        CONSTRAINT PK_alumno PRIMARY KEY(dni),
        CONSTRAINT FK_alumno_curso FOREIGN KEY(codCurso)
                REFERENCES cursos(codigo)
                        ON UPDATE CASCADE
                        ON DELETE NO ACTION);
```

Para consulta las tablas disponibles de una base de datos en MariaDB se utiliza el comando **SHOW TABLES**

```
mysql> SHOW TABLES;
+----------------------+
| Tables_in_prueba  |
+----------------------+
| alumnos                  |
| cursos                    |
+----------------------+

mysql>
```

## 2.1.4. TIPO DE DATOS

Podemos destacar los siguientes tipos de datos:

- **Numéricos enteros**

  Comencemos por conocer las opciones que tenemos para almacenar datos que sean numéricos enteros (edades, cantidades, magnitudes sin decimales); poseemos una variedad de opciones:

| Tipos de datos | Bytes | Valor mínimo | Valor máximo |
|---|---|---|---|
| TINYINT | 1 | -128 | 127 |
| SMALLINT | 2 | -32768 | 32767 |
| MEDIUMINT | 3 | -8388608 | 8388607 |
| INT | 4 | -2147483648 | 2147483647 |
| BIGINT | 8 | -9223372036854775808 | 9223372036854775808 |

- **Números decimales**.
  Estos tipos de datos son necesarios para almacenar precios, salarios, importes de cuentas bancarias, etc. que no son enteros.

  Tenemos que tener en cuenta que, si bien estos tipos de datos se llaman "de coma flotante", por ser la coma el separador entre la parte entera y la parte decimal, en realidad **MariaDB** los almacena usando un punto como separador. Por ejemplo,

  **DECIMAL(6,2)**. Tipo de datos decimal compuesto por 6 dígitos de los cuales 2 corresponden a la parte decimal.

- **Cadenas de caracteres.**
  El tipo de dato **VARCHAR** (character varying, o caracteres variables) es útil cuando la longitud del dato es desconocida, cuando depende de la información que el usuario escribe en campos o áreas de texto de un formulario.

  La longitud máxima permitida era de 255 caracteres hasta MySQL 5.0.3. pero desde esta versión cambio a un máximo de 65.535 caracteres.

- Datos de fecha y hora.
  - El tipo de dato **DATE** nos permite almacenar fechas en el formato: AAAA-MM-DD (los cuatro primeros dígitos para el año, los dos siguientes para el mes, y los ultimos dos para el dia).

    **Atención:** En los países de habla hispana estamos acostumbrados a ordenar las fechas en Día, Mes y Año, pero para MySQL y MariaDB es exactamente al revés.

  - Un campo definido como **DATETIME** nos permitirá almacenar información acerca de un instante de tiempo, pero no sólo la fecha sino también su horario, en el formato: AAAA-MM-DD HH:MM:SS

    Siendo la parte de la fecha de un rango similar al del tipo DATE (desde el 1000-01-01 00:00:00 al 9999-12-31 23:59:59), y la parte del horario, de 00:00:00 a 23:59:59.

  - **TIME.** Este tipo de cambio permite almacenar horas, minutos y segundos, en el formato HH:MM:SS, y su rango permitido va desde -839:59:59 hasta 839:59:59 (unos 35 días hacia atrás y hacia adelante de la fecha actual). Esto lo hace ideal para calcular tiempos transcurridos entre dos momentos cercanos.

## 2.1.5. MODIFICACIÓN DE TABLAS

El comando para modificar una tabla es ALTER TABLE y su sintaxis también es bastante compleja:

```
ALTER TABLE nombre_tabla
        ADD definición_columna [FIRST | AFTER nombre_columna]
|       ADD CONSTRAINT nombre_restricción PRIMARY  KEY
(nombre_columna, ...)
|       ADD CONSTRAINT nombre_restricción FOREIGN  KEY
(nombre_columna, ...)
                definición_referencia
|       MODIFY definición_columna [FIRST | AFTER nombre_columna]
|       DROP COLUMN nombre_columna
|       DROP PRIMARY KEY
|       DROP FOREIGN KEY fk_simbolo;
```

Al ver esta sintaxis se puede caer en la tentación de pensar que es mejor borrar una tabla y volver a crearla haciendo las modificaciones oportunas. Esto es posible si la tabla no tiene datos, pero ... ¿qué ocurre si hay un centenar o un millar de registros? No queda otra opción que ejecutar una sentencia **ALTER TABLE** para evitar la pérdida de datos.

- La opción **ADD** permite añadir una columna, se puede especificar el lugar donde se va a insertar mediante las cláusulas AFTER (después de una columna) y FIRST (la primera columna).
- Con la opción **MODIFY** se cambia el tipo de datos de una columna y se añaden restricciones.
- Con la opción de **DROP** se pueden eliminar las restricciones de claves foráneas y primarias, dejando el tipo de dato y su contenido intacto.

### 2.1.6. BORRADO DE TABLAS

El formato de instrucción de MySQL y MariaDB para eliminar una tabla es muy sencillo:

```
DROP TABLE [IF EXISTS] nombre_tabla
```

Por ejemplo:

```
DROP TABLE IF EXISTS alumnos;
```