Unidad Didáctica 4. Creación de bases de datos en MySQL

Apuntes de BD para DAW, DAM y ASIR

José Juan Sánchez Hernández

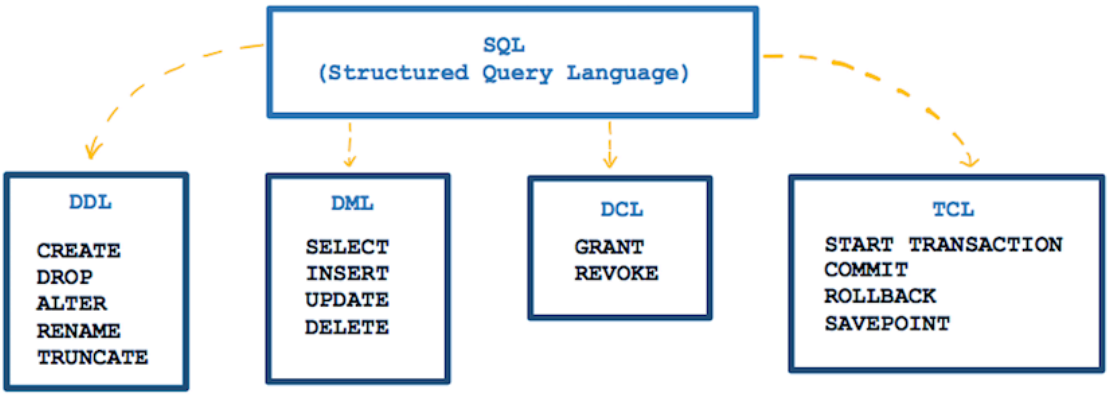
Curso 2025/2026

Índice

Índice de figuras

Índice de cuadros

1 El lenguaje DDL de SQL



El **DDL** (*Data Definition Language*) o **Lenguaje de Definición 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 que usaremos en esta unidad 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.

2 Manipulación de Bases de Datos

2.1 Crear una base de datos

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] nombre_base_datos;
```

- `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.

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 queremos de forma explícita:

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

2.1.1 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.

2.1.2 `utf8` y `utf8mb4` en MySQL

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`

En la documentación oficial de MySQL informan que en las próximas versione de de MySQL se espera solucionar este problema haciendo que `utf8` sea un alias de `utf8mb4`. Hasta que esto no ocurra, se recomienda utilizar `utf8mb4`.

2.1.3 `CHARACTER SET` y `COLLATE`

- `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.

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

```
1  SHOW CHARACTER SET;
```

Para consultar qué tipos de cotejamiento hay disponibles podemos usar:

```
1  SHOW COLLATION;
```

Si queremos hacer una consulta más específica sobre los tipos de cotejamiento que podemos usar con `utf8`:

```
1  SHOW COLLATION LIKE 'utf8%';
```

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.

Para consultar qué set de caracteres y qué cotejamiento está utilizando una determinada base de datos podemos consultar el valor de las variables `character_set_database` y `collation_database`.

En primer lugar seleccionamos la base de datos con la que vamos a trabajar.

```
USE database;
```

Y una vez seleccionada, consultamos el valor de las variables `character_set_database` y `collation_database` .

```
SELECT @@character_set_database, @@collation_database;
```

2.1.4 Ejemplo de cómo afecta el cotejamiento al ordenar una tabla

Suponemos que tenemos una tabla que contiene cadenas de caracteres codificadas en `latin1`.

```
mysql> CREATE TABLE t (c CHAR(3) CHARACTER SET latin1);

mysql> INSERT INTO t (c) VALUES('AAA'),('bbb'),('aaa'),('BBB');

mysql> SELECT c FROM t;
+------+
| c    |
+------+
| AAA  |
| bbb  |
| aaa  |
| BBB  |
+------+
```

Ahora vamos a obtener los registros de la tabla aplicando diferentes tipos de cotejamiento:

- Cotejamiento **case-sensitive** (los caracteres a y A son diferentes).

```
mysql> SELECT c FROM t ORDER BY c COLLATE latin1_general_cs;
+------+
| c    |
+------+
| AAA  |
| aaa  |
| BBB  |
| bbb  |
+------+
```

- Cotejamiento **case-insensitive** (los caracteres a y A son iguales).

```
mysql> SELECT c FROM t ORDER BY c COLLATE latin1_swedish_ci;
+------+
| c    |
+------+
```

```
| AAA  |
| aaa  |
| bbb  |
| BBB  |
+------+
```

- Cotejamiento **binary** (dos caracteres son iguales si los valores de su representación numérica son iguales).

```
mysql> SELECT c FROM t ORDER BY c COLLATE latin1_bin;
+------+
| c    |
+------+
| AAA  |
| BBB  |
| aaa  |
| bbb  |
+------+
```

2.2 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;
```

2.3 Modificar una base de datos

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

Ejemplo:

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

2.4 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.

2.5 Seleccionar una base de datos

```
USE nombre_base_datos;
```

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

2.6 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.

3 Manipulación de tablas

3.1 Crear una tabla

A continuación se muestra una **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] tbl_name
    (create_definition,...)
    [table_options]

create_definition:
    col_name column_definition
  | [CONSTRAINT [symbol]] PRIMARY KEY (index_col_name,...)
  | [CONSTRAINT [symbol]] FOREIGN KEY (index_col_name,...) reference_definition
  | CHECK (expr)

column_definition:
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]

reference_definition:
    REFERENCES tbl_name (index_col_name,...)
      [ON DELETE reference_option]
      [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options:
    table_option [[,] table_option] ...

table_option:
    AUTO_INCREMENT [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | ENGINE [=] engine_name
```

3.1.1 Restricciones sobre las columnas de la tabla

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

- `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.
- `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.
- `PRIMARY KEY`: Para indicar que una columna o varias son clave primaria.
- `CHECK`: Nos permite realizar restricciones sobre una columna. En las versiones previas a MySQL 8.0 estas restricciones no se aplicaban, sólo se parseaba la sintaxis pero eran ignoradas por el sistema gestor de base de datos. A partir de la versión de MySQL 8.0 ya sí se aplican las restricciones definidas con `CHECK`.

Ejemplo 1:

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

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

CREATE TABLE pieza (
  id 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),
  id_categoria INT UNSIGNED NOT NULL,
  FOREIGN KEY (id_categoria) REFERENCES categoria(id)
);
```

Ejemplo 2:

```sql
DROP DATABASE IF EXISTS agencia;
CREATE DATABASE agencia CHARSET utf8mb4;
USE agencia;

CREATE TABLE turista (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  apellidos VARCHAR(100) NOT NULL,
  direccion VARCHAR(100) NOT NULL,
  telefono VARCHAR(9) NOT NULL
);

CREATE TABLE hotel (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  direccion VARCHAR(100) NOT NULL,
  ciudad VARCHAR(25) NOT NULL,
  plazas INTEGER NOT NULL,
  telefono VARCHAR(9) NOT NULL
);

```

```sql
CREATE TABLE reserva (
  id_turista INT UNSIGNED NOT NULL,
  id_hotel INT UNSIGNED NOT NULL,
  fecha_entrada DATETIME NOT NULL,
  fecha_salida DATETIME NOT NULL,
  regimen ENUM('MP', 'PC'),
  PRIMARY KEY (id_turista,id_hotel),
  FOREIGN KEY (id_turista) REFERENCES turista(id),
  FOREIGN KEY (id_hotel) REFERENCES hotel(id)
);
```

3.1.2 Opciones en la declaración de claves ajenas (`FOREIGN KEY`)

- `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:
  - `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.
  - `NO ACTION`: Es una palabra clave del estándar SQL. En MySQL es equivalente a `RESTRICT`.
  - `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.

Ejemplo 1:

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

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

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

INSERT INTO categoria VALUES (1, 'Categoria A');
INSERT INTO categoria VALUES (2, 'Categoria B');
INSERT INTO categoria VALUES (3, 'Categoria C');

INSERT INTO pieza VALUES (1, 'Pieza 1', 'Blanco', 25.90, 1);
```

```
INSERT INTO pieza VALUES (2, 'Pieza 2', 'Verde', 32.75, 1);
INSERT INTO pieza VALUES (3, 'Pieza 3', 'Rojo', 12.00, 2);
INSERT INTO pieza VALUES (4, 'Pieza 4', 'Azul', 24.50, 2);
```

- ¿Podríamos borrar la **Categoría A** de la tabla **categoria**?
- ¿Y la **Categoría C**?
- ¿Podríamos actualizar la **Categoría A** de la tabla **categoria**?

Ejemplo 2:

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

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

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

INSERT INTO categoria VALUES (1, 'Categoria A');
INSERT INTO categoria VALUES (2, 'Categoria B');
INSERT INTO categoria VALUES (3, 'Categoria C');

INSERT INTO pieza VALUES (1, 'Pieza 1', 'Blanco', 25.90, 1);
INSERT INTO pieza VALUES (2, 'Pieza 2', 'Verde', 32.75, 1);
INSERT INTO pieza VALUES (3, 'Pieza 3', 'Rojo', 12.00, 2);
INSERT INTO pieza VALUES (4, 'Pieza 4', 'Azul', 24.50, 2);
```

- ¿Podríamos borrar la **Categoría A** de la tabla **categoria**?
- ¿Qué le ocurre a las piezas que pertenecen la **Categoría A** después de borrarla?
- ¿Podríamos actualizar la **Categoría A** de la tabla **categoria**?
- ¿Qué le ocurre a las piezas que pertenecen la **Categoría A** después de actualizarla?

Ejemplo 3:

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

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

```

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

INSERT INTO categoria VALUES (1, 'Categoria A');
INSERT INTO categoria VALUES (2, 'Categoria B');
INSERT INTO categoria VALUES (3, 'Categoria C');

INSERT INTO pieza VALUES (1, 'Pieza 1', 'Blanco', 25.90, 1);
INSERT INTO pieza VALUES (2, 'Pieza 2', 'Verde', 32.75, 1);
INSERT INTO pieza VALUES (3, 'Pieza 3', 'Rojo', 12.00, 2);
INSERT INTO pieza VALUES (4, 'Pieza 4', 'Azul', 24.50, 2);
```

- ¿Podríamos borrar la **Categoría A** de la tabla **categoria**?
- ¿Qué le ocurre a las piezas que pertenecen la **Categoría A** después de borrarla?
- ¿Podríamos actualizar la **Categoría A** de la tabla **categoria**?
- ¿Qué le ocurre a las piezas que pertenecen la **Categoría A** después de actualizarla?

3.1.3 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:

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

CREATE TABLE categoria (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
```

```
  nombre VARCHAR(100) NOT NULL
)  ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1000;

CREATE TABLE pieza (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  color VARCHAR(50) NOT NULL,
  precio DECIMAL(7,2) NOT NULL,
  id_categoria INT UNSIGNED NOT NULL,
  FOREIGN KEY (id_categoria) REFERENCES categoria(id)
)  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 inicial para las columnas de tipo **AUTO_INCREMENT**.

3.2 Eliminar una tabla

```
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;
```

3.3 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 necesaria para la modificación de una tabla en MySQL.

Puede consultar la sintaxis de modificación de tablas en la documentación oficial de MySQL.

```
ALTER TABLE tbl_name
    [alter_specification [, alter_specification] ...]
    [partition_options]

alter_specification:
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX|KEY} [index_name]
        [index_type] (index_col_name,...) [index_option] ...
```

```
    | ADD [CONSTRAINT [symbol]] PRIMARY KEY
          [index_type] (index_col_name,...) [index_option] ...
    | ADD [CONSTRAINT [symbol]]
          UNIQUE [INDEX|KEY] [index_name]
          [index_type] (index_col_name,...) [index_option] ...
    | ADD FULLTEXT [INDEX|KEY] [index_name]
          (index_col_name,...) [index_option] ...
    | ADD SPATIAL [INDEX|KEY] [index_name]
          (index_col_name,...) [index_option] ...
    | ADD [CONSTRAINT [symbol]]
          FOREIGN KEY [index_name] (index_col_name,...)
          reference_definition
    | ALGORITHM [=] {DEFAULT|INPLACE|COPY}
    | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
    | CHANGE [COLUMN] old_col_name new_col_name column_definition
          [FIRST|AFTER col_name]
    | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
    | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
    | {DISABLE|ENABLE} KEYS
    | {DISCARD|IMPORT} TABLESPACE
    | DROP [COLUMN] col_name
    | DROP {INDEX|KEY} index_name
    | DROP PRIMARY KEY
    | DROP FOREIGN KEY fk_symbol
    | FORCE
    | LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
    | MODIFY [COLUMN] col_name column_definition
          [FIRST | AFTER col_name]
    | ORDER BY col_name [, col_name] ...
    | RENAME {INDEX|KEY} old_index_name TO new_index_name
    | RENAME [TO|AS] new_tbl_name
    | {WITHOUT|WITH} VALIDATION
    | ADD PARTITION (partition_definition)
    | DROP PARTITION partition_names
    | DISCARD PARTITION {partition_names | ALL} TABLESPACE
    | IMPORT PARTITION {partition_names | ALL} TABLESPACE
    | TRUNCATE PARTITION {partition_names | ALL}
    | COALESCE PARTITION number
    | REORGANIZE PARTITION partition_names INTO (partition_definitions)
    | EXCHANGE PARTITION partition_name WITH TABLE tbl_name [{WITH|WITHOUT}
      VALIDATION]
    | ANALYZE PARTITION {partition_names | ALL}
    | CHECK PARTITION {partition_names | ALL}
    | OPTIMIZE PARTITION {partition_names | ALL}
    | REBUILD PARTITION {partition_names | ALL}
    | REPAIR PARTITION {partition_names | ALL}
    | REMOVE PARTITIONING
    | UPGRADE PARTITIONING

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option:
    KEY_BLOCK_SIZE [=] value
```

```
    | index_type
    | WITH PARSER parser_name
    | COMMENT 'string'

table_options:
    table_option [[,] table_option] ...

table_option:
    AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | COMPRESSION [=] {'ZLIB'|'LZ4'|'NONE'}
  | CONNECTION [=] 'connect_string'
  | {DATA|INDEX} DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | ENCRYPTION [=] {'Y' | 'N'}
  | ENGINE [=] engine_name
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT|DYNAMIC|FIXED|COMPRESSED|REDUNDANT|COMPACT}
  | STATS_AUTO_RECALC [=] {DEFAULT|0|1}
  | STATS_PERSISTENT [=] {DEFAULT|0|1}
  | STATS_SAMPLE_PAGES [=] value
  | TABLESPACE tablespace_name [STORAGE {DISK|MEMORY|DEFAULT}]
  | UNION [=] (tbl_name[,tbl_name]...)

partition_options:
    (see CREATE TABLE options)
```

3.3.1 Ejemplo de `ALTER TABLE <tbl_name> MODIFY`

`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:

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

3.3.2 Ejemplo de ALTER TABLE <tbl_name> CHANGE

`CHANGE` nos permite renombrar una columna, modificar el tipo de dato de una columna y sus atributos.

Suponemos que tenemos la siguiente tabla creada

```sql
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:

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

Después de ejecutar esta sentencia la tabla quedaría así:

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

3.3.3 Ejemplo de ALTER TABLE <tbl_name> ALTER

`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

```sql
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  rol ENUM('Estudiante', 'Profesor') NOT NULL
);
```

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

```sql
ALTER TABLE usuario ALTER rol SET DEFAULT 'Estudiante';
```

Después de ejecutar esta sentencia la tabla quedaría así:

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

```
  rol ENUM('Estudiante', 'Profesor') NOT NULL DEFAULT 'Estudiante'
);
```

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

```
ALTER TABLE usuario ALTER rol DROP DEFAULT;
```

3.3.4 Ejemplo de `ALTER TABLE <tbl_name> ADD`

`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,
  rol ENUM('Estudiante', 'Profesor') NOT NULL
);
```

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

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

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

```
CREATE TABLE usuario (
  id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(50) NOT NULL,
  rol ENUM('Estudiante', 'Profesor') 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 apellido1 VARCHAR(50) NOT NULL AFTER nombre;

ALTER TABLE usuario ADD 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),
  rol ENUM('Estudiante', 'Profesor') NOT NULL DEFAULT 'Estudiante',
  fecha_nacimiento DATE NOT NULL
);
```

3.3.5 Ejemplo de `ALTER TABLE <tbl_name> DROP`

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),
  rol ENUM('Estudiante', 'Profesor') NOT NULL DEFAULT 'Estudiante',
  fecha_nacimiento DATE NOT NULL
);
```

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

```
ALTER TABLE usuario DROP 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),
  rol ENUM('Estudiante', 'Profesor') NOT NULL DEFAULT 'Estudiante'
);
```

3.4 Consultar el listado de tablas disponibles

```
SHOW TABLES;
```

Muestra un listado de todas las tablas que existen en la base de datos con la que estamos trabajando.

3.5 Mostrar información sobre la estructura de una tabla

```
DESCRIBE nombre_tabla;
```

También podemos utilizar:

```
DESC nombre_tabla;
```

Esta sentencia se utiliza para mostrar información sobre la estructura de una tabla.

3.6 Mostrar la sentencia SQL de creación de una tabla

```sql
SHOW CREATE TABLE nombre_tabla;
```

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

4 Tipos de datos

4.1 Números enteros

Tipo	Bytes	Mínimo	Máximo
`TINYINT`	1	-128	127
`TINYINT UNSIGNED`	1	0	255
`SMALLINT`	2	-32768	32767
`SMALLINT UNSIGNED`	2	0	65535
`MEDIUMINT`	3	-8388608	8388607
`MEDIUMINT UNSIGNED`	3	0	16777215
`INT`	4	-2147483648	2147483647
`INT UNSIGNED`	4	0	4294967295
`INTEGER`	4	-2147483648	2147483647
`INTEGER UNSIGNED`	4	0	4294967295
`BIGINT`	8	-9223372036854775808	9223372036854775807
`BIGINT UNSIGNED`	8	0	18446744073709551615

Puedes encontrar más información sobre números enteros en la documentación oficial de MySQL.

4.1.1 `ZEROFILL`

Todos los tipos de datos numéricos admiten el atributo `ZEROFILL`. Cuando asignamos este atributo a una columna también se le añade de forma automática el atributo `UNSIGNED`, de modo que el campo quedaría como `UNSIGNED ZEROFILL`.

4.1.2 Nota importante sobre INT(11)

`INT(11)` **no** quiere decir que queremos guardar un número entero de 11 cifras. El número indicado entre paréntesis **indica el ancho de la columna que ocupará dicho valor** y tiene utilidad cuando asignamos el atri-

buto **UNSIGNED ZEROFILL**. En este caso se completa con 0 a la izquierda del valor hasta alcanzar el número indicado entre paréntesis.

Por ejemplo, para una columna declarada como `INT(4) ZEROFILL`, el valor 5 será representado como 0005.

4.1.3 BIT, BOOL, BOOLEAN, SERIAL

Tipo	Descripción
BIT(M)	M puede ser un valor de 1 a 64. Indica el número de bits que vamos a utilizar para este campo. Si se omite el valor de M se utiliza 1 bit por defecto.
BOOL, BOOLEAN	Son equivalentes a `TINYINT(1)`. El valor 0 se considera como `FALSE`. Cualquier valor distinto de 0 será `TRUE`.
SERIAL	Es un alias para: `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

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

4.2 Números en punto flotante (Valores aproximados)

Los tipos de datos `FLOAT` y `DOUBLE` almacenan valores numéricos aproximados y no valores exactos, por lo tanto, debe tener en cuenta que podemos obtener resultados erróneos a la hora de realizar comparaciones exactas. En la documentación oficial de MySQL podemos encontrar un ejemplo de los problemas que podemos tener con los valores en punto flotante.

Tipo	Bytes	Mínimo	Máximo
FLOAT	4		
FLOAT(M,D)	4	±1.175494351E-38	±3.402823466E+38
FLOAT(M,D) UNSIGNED	4	1.175494351E-38	3.402823466E+38
DOUBLE	8		
DOUBLE(M,D)	8	±1.7976931348623157E+308	±2.2250738585072014E-308

Tipo	Bytes	Mínimo	Máximo
`DOUBLE(M,D) UNSIGNED`	8	1.7976931348623157E+308	2.2250738585072014E-308

- `M` indica el número de dígitos en total (la precisión).
- `D` es el número de cifras decimales.

MySQL permite utilizar una sintaxis no estándar para definir los tipos de datos `FLOAT` y `DOUBLE` como: `FLOAT(M,D)` y `DOUBLE(M,D)`. Donde `(M,D)` representan que los valores pueden ser almacenados con `M` dígitos en total (parte entera más parte decimal), de los cuales `D` dígitos serán para la parte decimal. A partir de la versión 8.0.17 de MySQL esta sintaxis está obsoleta.

Por ejemplo, un número declarado como `FLOAT(7,4)` tendrá 7 dígitos como máximo y 4 de ellos serán decimales. El rango de números que se pueden representar en este caso será desde `-999.9999` hasta `999.9999`.

El estándar SQL permite indicar de forma opcional el número de bits (precisión) que se van a utilizar para almacenar la mantisa en los datos de tipo `FLOAT`. En este caso, el número de bits (precisión) se indicará entre paréntesis a continuación de la palabra reservada `FLOAT`, por ejemplo: `FLOAT(p)`, donde `p` indica el número de bits de la mantisa.

El tipo de dato `FLOAT` representa un número real de 32 bits en simple precisión, con 1 bit para el signo, 8 bits para el exponente y 23 para la mantisa, por lo tanto, a la hora de definir una precisión para este tipo de dato podremos utilizar un valor entre 0 y 23 bits.

Puedes encontrar más información sobre números en punto flotante en la documentación oficial de MySQL.

4.2.1 Problemas de precisión con operaciones en punto flotante

A continuación, se muestran algunas referencias que pueden útiles para comprender los problemas de precisión que pueden aparecer en las operaciones con datos en punto flotante.

- Floating-point Accuracy. MariaDB.
- Accuracy problems. Wikipedia.
- Lo que todo programador debería saber sobre aritmética de punto flotante.
- IEEE-754 Floating Point Converter.
- Transparencias sobre representación de la información. Grupo ARCOS. Alejandro Calderón Mateos. UC3M.

4.3 Números en punto fijo (Valores exactos)

Los tipos de datos `DECIMAL` y `NUMERIC` almacenan valores numéricos exactos y se utilizan cuando es necesario guardar los valores exactos sin redondeos. Se suelen utilizar cuando trabajamos con **datos monetarios**.

En la documentación oficial de MySQL puede encontrar información sobre cómo MySQL proporciona soporte para realizar operaciones matemáticas con precisión.

Tipo	Bytes
DECIMAL	
DECIMAL(M,D)	M + 2 bytes si D > 0
DECIMAL(M,D)UNSIGNED	M + 1 bytes si D = 0
NUMERIC	
NUMERIC(M,D)	M + 2 bytes si D > 0
NUMERIC(M,D)UNSIGNED	M + 1 bytes si D = 0

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.

Por ejemplo, un número declarado como DECIMAL(7,4) tendrá 7 dígitos como máximo y 4 de ellos serán decimales. El rango de números que se pueden representar en este caso será desde -999.9999 hasta 999.9999 .

Cuando se declara una columna como DECIMAL y no se indica la precisión (M) ni el número de cifras decimales (D), se utilizarán los valores por defecto, que es equivalente a declarar la columna como DECIMAL(10,0).

Si sólo se indica la precisión (M) y no se indica el número de cifras decimales (D), entonces la columna no almacenará decimales. Por ejemplo, si declaramos una columna como DECIMAL(7), es equivalente a declararla como DECIMAL(7,0).

Puedes encontrar más información sobre números en punto fijo en la documentación oficial de MySQL.

4.4 Fechas y tiempo

Tipo	Bytes	Descripción	Rango	Máximo
DATE	3	YYYY-MM-DD	1000-01-01	9999-12-31
DATETIME	8	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00	9999-12-31 23:59:59
TIMESTAMP	4	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:00	2038-01-19 03:14:07
TIME	3	HH:MM:SS	-838:59:59	838:59:59
YEAR[(2|4)]	1	YY o YYYY	YY: 70 (1970)	YY: 69 (2069)

Tipo	Bytes	Descripción	Rango	Máximo
			YYYY: 1901	YYYY: 2155

Puedes encontrar más información sobre fechas y tiempo en la documentación oficial de MySQL.

Se recomienda la lectura del artículo que hay en Wikipedia sobre el problema del año 2038.

4.5 Cadenas de caracteres

Tipo	Descripción
`CHAR(M)`	0 <= M <= 255
`VARCHAR(M)`	0 <= M <= 65535
`TEXT[(M)]`	L + 2 bytes, donde L < 216 = 65536
`MEDIUMTEXT`	L + 3 bytes, donde L < 224 = 16 MB
`LONGTEXT`	L + 4 bytes, donde L < 232 = 4 GB

Puedes encontrar más información sobre cadenas de caracteres en la documentación oficial de MySQL.

4.6 Datos binarios

Tipo	Descripción
`BINARY`	0 <= M <= 255
`VARBINARY`	0 <= M <= 65535
`BLOB`	L + 2 bytes, donde L < 216 = 65536
`MEDIUMBLOB`	L + 3 bytes, donde L < 224 = 16 MB
`LONGBLOB`	L + 4 bytes, donde L < 232 = 4 GB

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

4.7 ENUM y SET

Tipo	Descripción
`ENUM('valor1', 'valor2', ...)`	Puede tener 65535 valores. Sólo permite seleccionar un valor de la lista
`SET('valor1', 'valor2', ...)`	Puede tener 64 valores. Permite seleccionar varios valores de la lista

Puedes encontrar más información sobre estos tipos de datos en la documentación oficial de MySQL (ENUM y SET).

4.8 JSON

Tipo	Descripción
`JSON`	Documentos JSON (JavaScript Object Notation)

MySQL también incluye un tipo de dato específico para almacenar datos en formato JSON (JavaScript Object Notation). El formato JSON está definido en el RFC 7159 y se trata de un formato de texto para el intercambio de datos.

Las ventajas de utilizar el tipo de dato JSON en lugar de una cadena de caracteres (VARCHAR, TEXT, etc.) son:

- Realiza una validación automática de la sintaxis del documento JSON que se quiere almacenar y no permite almacenar documentos que contengan errores de sintaxis.
- Los documentos se almacenan en un formato binario optimizado que permiten acceder a los elementos del documento de una forma más eficiente.

Puedes encontrar más información sobre este tipo de dato en la documentación oficial de MySQL.

Ejemplo:

```sql
DROP DATABASE IF EXISTS ejemplo;
CREATE DATABASE ejemplo CHARSET utf8mb4;
USE ejemplo;

CREATE TABLE tabla (
  documento JSON
);

INSERT INTO tabla VALUES('{"key1": "value1", "key2": "value2"}');
INSERT INTO tabla VALUES (JSON_OBJECT('key1', 1, 'key2', '2'));

SELECT * FROM tabla;

SELECT documento->"$.key1" FROM tabla;
SELECT documento->"$.key2" FROM tabla;
```

4.9 Resumen de los tipos de datos disponibles en MySQL

```
BIT[(length)]
TINYINT[(length)] [UNSIGNED] [ZEROFILL]
SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
INT[(length)] [UNSIGNED] [ZEROFILL]
INTEGER[(length)] [UNSIGNED] [ZEROFILL]
BIGINT[(length)] [UNSIGNED] [ZEROFILL]
REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
DECIMAL[(length[,decimals])] [UNSIGNED] [ZEROFILL]
NUMERIC[(length[,decimals])] [UNSIGNED] [ZEROFILL]
DATE
TIME
TIMESTAMP
DATETIME
YEAR
CHAR[(length)] [BINARY] [CHARACTER SET charset_name] [COLLATE collation_name]
VARCHAR(length) [BINARY] [CHARACTER SET charset_name] [COLLATE collation_name]
BINARY[(length)]
VARBINARY(length)
TINYBLOB
BLOB
MEDIUMBLOB
LONGBLOB
TINYTEXT [BINARY] [CHARACTER SET charset_name] [COLLATE collation_name]
TEXT [BINARY] [CHARACTER SET charset_name] [COLLATE collation_name]
MEDIUMTEXT [BINARY] [CHARACTER SET charset_name] [COLLATE collation_name]
LONGTEXT [BINARY] [CHARACTER SET charset_name] [COLLATE collation_name]
ENUM(value1,value2,value3,...) [CHARACTER SET charset_name] [COLLATE
    collation_name]
SET(value1,value2,value3,...) [CHARACTER SET charset_name] [COLLATE
    collation_name]
spatial_type
JSON
```

4.10 Valores fuera de rango y desbordamientos (*Out-of-Range and Overflow*)

Cuando MySQL almacena en una columna de tipo numérico un valor que está fuera del rango permitido, pueden ocurrir dos situaciones que dependen de la configuración de MySQL (`sql_mode`).

- Si está habilitado el modo estricto, MySQL no permite que se inserten los valores que están fuera de rango y lanza un mensaje de error.

  Ejemplo.

  Si tenemos una columna con un tipo de dato `TINYINT UNSIGNED`, el rango de valores permitido para esta columna será `[0, 255]`. Si quisiéramos almacenar el valor `256` en esta columna MySQL lanzaría

un mensaje de error con el código 1264.

```
SET sql_mode = 'TRADITIONAL';

CREATE TABLE test (data TINYINT UNSIGNED);

INSERT INTO test VALUES(256);

ERROR 1264 (22003): Out of range value for column 'data' at row 1
```

```
Si consultamos el contenido de la tabla veremos que no se ha insertado ningún
    valor.
```

```
SELECT * FROM test;

Empty set (0.00 sec)
```

- Si no está habilitado el modo estricto, MySQL permite se que inserten los valores que están fuera de rango pero se adaptan al rango de valores que permita la columna.

  Ejemplo.

  Si tenemos una columna con un tipo de dato `TINYINT UNSIGNED`, el rango de valores permitido para esta columna será `[0, 255]`. Si quisiéramos almacenar el valor `400` en esta columna y MySQL está configurado en modo no estricto, se almacenaría `255` que es el máximo valor que se puede representar con este tipo de dato.

```
SET sql_mode = '';

CREATE TABLE test (data TINYINT UNSIGNED);

INSERT INTO test VALUES(400);

1 row(s) affected, 1 warning(s): 1264 Out of range value for column 'data' at
    row 1
```

```
Si consultamos el contenido de la tabla veremos que en lugar de almacenar el
valor `400` se ha almacenado el valor `255`.
```

```
SELECT * FROM test;
+------+
| data |
+------+
| 255  |
+------+
```

Puedes encontrar más información sobre como se gestionan los valores fuera de rango y el desbordamiento en la documentación oficial de MySQL.

4.10.1 Cómo configurar la variable `sql_mode`

Esta variable se puede configurar a nivel global o a nivel de sesión.

```
SET GLOBAL sql_mode = 'modes';
SET SESSION sql_mode = 'modes';
```

Donde algunos de los valores más importantes que podemos utilizar en `modes` son:

- `ANSI`
- `STRICT_TRANS_TABLES`
- `TRADITIONAL`

Puede consultar cuál es la lista de todos los modos que podemos utilizar en la documentación oficial.

4.10.2 Cómo consultar la variable `sql_mode`

Para consultar cuál es el valor que tienen estas variables podemos hacerlo así:

```
SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

Puede consultar más detalles sobre los modos de SQL en la documentación oficial.

4.11 Problemas de precisión con operaciones en punto flotante

- Floating-point Accuracy. MariaDB.
- Accuracy problems. Wikipedia.
- Lo que todo programador debería saber sobre aritmética de punto flotante.
- IEEE-754 Floating Point Converter.
- Transparencias sobre representación de la información. Grupo ARCOS. Alejandro Calderón Mateos. UC3M.

5 Índices

El uso de índices lo estudiaremos más adelante, en la unidad 10: “Optimización de consultas”.

Referencias:

- Unidad 10: Optimización de consultas.
- Documentación oficial de MySQL sobre la creación de índices.

6 Vistas

El uso de índices lo estudiaremos más adelante, en la unidad: “Vistas”.

Referencia:

- Unidad: Vistas.
- Documentación oficial de MySQL sobre la creación de vistas.

7 Referencias

- MySQL 8.0 Reference Manual.
- **Gestión de Bases de Datos**. 2ª Edición. Ra-Ma. Luis Hueso Ibáñez.
- **Bases de Datos**. 2ª Edición. Grupo editorial Garceta. Iván López Montalbán, Manuel de Castro Vázquez y John Ospino Rivas.
- Codificación de caracteres: conceptos básicos.
- The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding).
- **MySQL Cookbook**. Paul Dubois.
- 9 Reasons Why There Are No Foreign Keys in Your Database (Referential Integrity Checks). Piotr Kononow.

8 Licencia