Tema 4 - SQL

Antes de empezar hay que tener en cuenta varias características del lenguaje **SQL**:

- El lenguaje es **non-case-sensitive** (no sensible a mayús/minus)
- En el **DDL** la palabra reservada **DATABASE** también se puede usar como **SCHEMA**

Tipos de Datos

- **Tipo Numérico**

Tipo de dato	Descripción
TINYINT (tamaño)	-128 to 127 normal. 0 to 255 UNSIGNED*
SMALLINT (tamaño)	-32768 to 32767 normal. 0 to 65535 UNSIGNED*
MEDIUMINT (tamaño)	-8388608 to 8388607 normal. 0 to 16777215 UNSIGNED*.
INT (tamaño)	-2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED*.
BIGINT (tamaño)	-9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED*.
FLOAT (i,d)	Número pequeño con coma flotante. `i - Números izqda, d - Números dcha.`
DOUBLE (i,d)	Número grande con coma flotante. `i - Números izqda, d - Números dcha.`
DECIMAL (i,d)	Como un ***DOUBLE*** pero guardado en forma de STRING. `i - Números izqda, d - Números dcha.`

- **Tipo Texto**

Tipo de dato	Descripción
CHAR (tamaño)	Hasta 255 caracteres (letras,números y caracteres especiales)
VARCHAR (tamaño)	Hasta 255 caracteres (letras, números y caracteres especiales). Si el tamaño excede de 255 se convierte en tipo ***TEXT***
TINYTEXT	String de hasta 255 caracteres
TEXT	String de hasta 65.535 caracteres
BLOB	(Binary Large Object). Hasta 65.535 bytes de datos
MEDIUMTEXT	String de hasta 16.771.215 caracteres
MEDIUMBLOB	(Binary Large Object). Hasta 16.777.215 bytes de datos
LONGTEXT	String de hasta 4.294.967.295 caracteres
LONGBLOB	(Binary Large Object). Hasta 4.294.967.295 de datos
ENUM (x,y,z,etc.)	Hasta 65.535 valores en la lista
SET	Parecido a ***ENUM*** pero ***SET*** contiene hasta 64 valores en la lista y puede almacenar más de una elección

- **Tipo Fecha**

Tipo de dato	Descripción
DATE	Formato: `AAAA-MM-DD` Rango: 1000-01-01 hasta 9999-12-31
DATETIME	Formato: `AAAA-MM-DD HH:MM:SS` Rango: `1000-01-01 00:00:00` hasta `9999-12-31 23:59:59`
TIMESTAMP	Formato: `AAAA-MM-DD HH:MM:SS` Rango: `1970-01-01 00:00:01 UTC` hasta `2038-01-09 03:14:07 UTC`
TIME	Formato: `HH:MM:SS` Rango: `-838:59:59` hasta `838:59:59`
YEAR	Rango: Con 4 cifras: `1901` hasta `2155` Con 2 cifras: `70` hasta `69` haciendo referencia a `1970` y `2069`

Durante la explicación de los comandos SQL vamos a estar utilizando el mismo ejemplo de la tabla `PIZZAS` para una mejor comprensión. Utilizaremos la **sintaxis** de MariaDB / MySQL.

DDL

DDL viene de **Data Definition Language**, por tanto, sirve para definir datos.

Creación y modificación de datos

- ***Creación de la base de datos***

```
-- Sintaxis 1 --
CREATE DATABASE (IF NOT EXISTS) NOMBRE;
-- Sintaxis 2 --
CREATE SCHEMA (IF NOT EXISTS) NOMBRE;
```

```
-- Ejemplo 1 --
CREATE DATABASE IF NOT EXISTS PIZZERIA;
-- Ejemplo 2 --
CREATE SCHEMA IF NOT EXISTS PIZZERIA;
```

- ***Eliminación de la base de datos***

```
-- Sintaxis --
DROP DATABASE (IF EXISTS) NOMBRE;
```

```
-- Ejemplo --
DROP DATABASE IF EXISTS PIZZERIA;
```

- ***Selección de la base de datos***

```
-- Sintaxis --
USE NOMBRE_BD;
```

```
-- Ejemplo --
USE PIZZERIA;
```

- ***Creación de tabla***

```
-- Sintaxis --
CREATE TABLE (IF NOT EXISTS) NOMBRE_BD NOMBRE_TABLA (
nombre_columna tipo_de_datos [lista_de_restricciones],
...
);
```

```
-- Ejemplo --
CREATE TABLE IF NOT EXISTS PIZZERIA PIZZAS(
  idPizza INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  price DECIMAL(3,1) NOT NULL,
  date_creation DATE
);
```

- ***Añadir una nueva columna a una tabla***

```
-- Sintaxis --
ALTER TABLE [NOMBRE_BD] NOMBRE_TABLA
ADD NOMBRE_COLUMNA TIPO_DE_DATO [LISTA_DE_RESTRICCIONES];
```

```
-- Ejemplo --
ALTER TABLE PIZZAS
ADD special_offer DECIMAL(2,1)
```

- ***Añadir una restricción a una columna***

```
-- Sintaxis --
ALTER TABLE [NOMBRE_BD] NOMBRE_TABLA
ADD [CONSTRAINT] TIPO_RESTRICCION [(NOMBRE_COLUMNA)];
```

```
-- Ejemplo --
ALTER TABLE PIZZAS
ADD CONSTRAINT NOT NULL (date_creation);
```

- ***Eliminar una columna***

```
-- Sintaxis --
ALTER TABLE [NOMBRE_BD] NOMBRE_TABLA
DROP [COLUMN] NOMBRE_COLUMNA;
```

```
-- Ejemplo --
ALTER TABLE PIZZAS
DROP COLUMN date_creation;
```

- ***Eliminar la restricción de la clave principal***

```
-- Sintaxis --
ALTER TABLE [NOMBRE_BD] NOMBRE_TABLA
DROP PRIMARY KEY;
```

```
-- Ejemplo --
ALTER TABLE PIZZAS
DROP PRIMARY KEY;
```

- ***Cambiar características de una columna***

```
-- Sintaxis --
ALTER TABLE [NOMBRE_BD] NOMBRE_TABLA
MODIFY [COLUMN] NOMBRE_COLUMNA NUEVA_DEFINICION;
```

```
-- Ejemplo --
ALTER TABLE PIZZAS
MODIFY COLUMN price DECIMAL(4,2) NOT NULL;
```

- ***Cambiar el nombre de una columna***

```
-- Sintaxis --
ALTER TABLE [NOMBRE_BD] NOMBRE_TABLA
CHANGE [COLUMN] NOMBRE_ANTIGUO NOMBRE_NUEVO NUEVA_DEFINICION;
```

```
-- Ejemplo --
ALTER TABLE PIZZAS
CHANGE COLUMN price precio DECIMAL(6,2) NOT NULL;
```

- ***Cambiar el nombre de una tabla***

```
-- Sintaxis --
ALTER TABLE [NOMBRE_BD] NOMBRE_TABLA
RENAME TO NUEVO_NOMBRE;
```

```
-- Ejemplo --
ALTER TABLE PIZZAS
RENAME TO PIZZA_LIST;
```

- ***Eliminar una tabla***

```
-- Sintaxis --
DROP TABLE [IF EXISTS] [NOMBRE_BD] NOMBRE_TABLA;
```

```
-- Ejemplo --
DROP TABLE IF EXISTS PIZZAS;
```

Tipos de Restricciones

Tipo de Restriccion	Descripción
`CHECK`	Limita el rango de un valor
`NOT NULL`	Requiere que un valor no sea nulo
`UNIQUE`	Comprueba que todos los valores en una columna son diferentes al que se le asigna
`PRIMARY KEY`	Clave primaria de la tabla
`AUTO_INCREMENT`	Auto incrementa el valor automáticamente y tiene que ser `PRIMARY KEY`
`COMMENT`	Comenta algo sobre el campo definido
`FOREIGN KEY`	Clave extranjera Sintaxis: `FOREIGN KEY (COLUMNA) REFERENCES TABLA (OTRA_COLUMNA)`

DML

DML viene de **Data Modification Language**, por tanto, sirve para modificar datos ya creados o ya existentes, o bien agregar nuevos.

Insertar datos en una tabla

A la hora de insertar datos en una tabla **tenemos que tener en cuenta el tipo de dato con el que estamos trabajando**, ya que, por ejemplo, poner un `INT` entre comillas simples o poner el formato de la fecha mal nos va a dar error.

Para insertar datos nuevos en una tabla utilizamos la sentencia `INSERT`. Esta tiene dos formas de uso, la primera sería mediante la cláusula `VALUES`:

```
-- Sintaxis --
INSERT INTO NOMBRE_TABLA [(COLUMNA1,COLUMNA2,...)]
VALUES (VALOR1,VALOR2,...);
```

```
-- Ejemplo --.
INSERT INTO PIZZAS (idPizza,name,price,date_creation)
VALUES (12,'Carbonara',6.5,'2018-3-1');
```

*Tenemos que saber que los tipos de datos de fechas y de texto van entre comillas simples, mientras que los números no llevan **menos** el de tipo* `DECIMAL` .

La otra forma sería mediante la sentencia `SELECT`. Con esta sentencia tenemos que seleccionar columnas ya existentes, porque con este método realmente no insertamos nuevos valores del todo, si no que cogemos valores de otra columna ya existentes y los copiamos:

```
-- Sintaxis --
INSERT INTO NOMBRE_TABLA [(COLUMNA1,COLUMNA2,...)]
SELECT ... ;
```

```
-- Ejemplo --
INSERT INTO PIZZAS (idPizza,name,price,date_creation)
SELECT idShop
FROM SHOPS;
```

De esta forma hemos copiado la columna `idShop` de la tabla `SHOPS` a la tabla `PIZZAS`.

Actualizar datos en una tabla

A la hora de actualizar datos en una tabla, podemos recurrir a la cláusula `WHERE`, el cual sirve para seleccionar solo las columnas que cumplan una **condición**. Si no utilizamos esta sentencia, seleccionará todas las columnas.

```
-- Sintaxis --
UPDATE NOMBRE_TABLA
SET COLUMNA1=VALOR1, COLUMNA2=VALOR2,...
WHERE CONDICIÓN;
```

```
-- Ejemplo --
UPDATE PIZZAS
SET price=3.5
WHERE name LIKE 'Carbonara';
```

Como podemos comprobar, esta cláusula es **muy potente** y tiene muchas más opciones que las vistas en el ejemplo anterior. Vamos a proceder a ver todas las sintaxis posibles.

Condicionales/Operadores básicos:

Vamos a realizar ejemplos con los operadores más básicos de SQL enlazados con el `WHERE` para **seleccionar los valores que queramos**. En los ejemplos faltaría seleccionar la tabla con la que trabajamos ya que depende el caso se haría con el `UPDATE` o bien con el `SELECT` seguido del `FROM`.

- `=`

  Igual que.
- `<>` ó `!=`

  En algunas versiones de SQL es `!=` y en otras `<>` .
- `>` `<`

  Menor o mayor que.
- `<=` `>=`

  Menor o igual que y mayor o igual que.
- `AND` `OR` `NOT`

Operadores lógicos que actuan como el `&&` , `||` o el `!=`.

- `BETWEEN`

  Este operador nos permite seleccionar valores en un rango.

```
-- Sintaxis --
WHERE NOMBRE_COLUMNA BETWEEN VALOR1 AND VALOR2;
```

```
-- Ejemplo --
WHERE price BETWEEN 2 AND 4;
```

- `LIKE`

  En el operador `LIKE` tenemos muchas sintaxis diferentes a nuestro alcance.

```
-- Ejemplo simple --
WHERE name LIKE 'Carbonara'; -- Que el campo 'name' es 'Carbonara'

-- Ejemplos con el % --
WHERE name LIKE 'c%'; -- Que empieza con la letra 'c'
WHERE name LIKE '%c'; -- Que acaba con la letra 'c'
WHERE name LIKE '%pizza%'; -- Que contiene el string 'pizza'
WHERE name LIKE 'p%a'; -- Que empieza con la 'p' y acaba con la 'a'

-- También podríamos hacer la inversa utilizando el NOT LIKE:
WHERE name NOT LIKE 'Carbonara';
```

- `IN`

  Este operador nos permite especificar multiples valores, de manera que comprueba si un valor está dentro de esa 'lista'.

```
-- Sintaxis --
WHERE NOMBRE_COLUMNA IN (VALOR1, VALOR2, ...);
```

```
-- Ejemplo --
WHERE name IN ('Carbonara','Barbacoa','Tropical');
```

Eliminar datos en una tabla

El caso más práctico sería eliminar todos los datos de una tabla que cumplan una condición. Si lo hacemos sin la cláusula `WHERE` eliminaría todo.

```
-- Sintaxis --
DELETE FROM NOMBRE_TABLA
WHERE CONDICION;
```

```
-- Ejemplo --
DELETE FROM PIZZAS
WHERE price > 10;
```