SQL

- Lenguaje que permite consultar y modificar la información almacenada en una **base de datos**.

Existen 2 tipos de comandos dependiendo de cuál sea su función:

- DDL **(Data Definition Language).**
- DML **(Data Manipulation Language)**.

Tipos de datos:

- **Tipo numérico:** pueden tener valores decimales **(coma flotante)** o no.

    - BIT (1 byte por cada 8 bits).
    - TINYINT (1 byte de espacio).
    - SMALLINT (2 bytes de espacio).
    - INT (4 bytes de espacio).
    - BIGINT (8 bytes de espacio).
    - FLOAT (4 bytes de espacio).
    - DOUBLE (8 bytes de espacio).

- **Tipo fecha:** para representar **fechas**.

    - **DATE:** almacena una **fecha (año-mes-día)**.
    - **TIME:** almacena una **hora**, **minutos** y **segundos (horas:minutos:segundos)**.
    - **DATETIME:** combinación de **fecha y hora (año-mes-día horas:minutos:segundos)**.

- **Tipo carácter:** para representar **caracteres** o **cadenas de caracteres**.

    - **CHAR(n):** almacena una cadena de longitud fija n, que podrá tener valores entre 0 y 255.
    - **VARCHAR(n):** almacena una cadena de longitud variable, siendo n el máximo de caracteres de la cadena. n tomará valor entre 0 y 255.

Creación y eliminación de la base de datos.

- **Create Database:** crea una base de datos. Ej: **CREATE DATABASE nuevadb;**
- **Drop Database:** borra una base de datos. Ej: **DROP DATABASE nuevadb;**

Uso de la base de datos.

- **Use:** selecciona para usar una base de datos. Ej: **USE nuevadb;**

DDL. Operaciones sobre bases de datos en SQL.

Permiten crear nuevas bases de datos, añadiendo y eliminando elementos.

- **Create Table:** crea una nueva tabla.
  Ej: **CREATE TABLE nuevaTabla;**

- **Alter Table:** agrega, modifica o elimina columnas de una tabla.
  Ej: **ALTER TABLE nuevaTabla;**

    - **ALTER TABLE:** agregar columnas (ADD/ADD RESTRICCION).
      Ej: **ALTER TABLE nuevaTabla ADD nuevaColumn char (255);**

    - **ALTER TABLE:** eliminar columnas **(DROP COLUMN/DROP RESTRICCION)**.
      Ej: **ALTER TABLE nuevaTabla DROP COLUMN nuevaColumn;**

- **Drop Table:** elimina tablas.
  Ej: **DROP TABLE nuevaTabla;**

- **Truncate Table:** borra el contenido de una tabla, pero no la tabla como tal.
  Ej: **TRUNCATE TABLE nuevaTabla;**

Restricciones sobre las columnas:

- **Not Null:** la columna debe recibir un valor en la creación o la modificación.
  Ej: **columnaTabla VARCHAR (255) NOT NULL**.

- **Default:** fuerza un valor si no se ha precisado en la creación del registro.
  Ej: **columnaTabla VARCHAR (255) DEFAULT**.

- **Auto Increment:** en la creación o en la modificación de una tabla, permite que el sistema genere valores automáticamente para columna numérica entera.
  Ej: **id_tabla INT AUTO_INCREMENT**.

- **Primary Key:** permite definir un identificador de clave primaria para la tabla **(solo puede haber una)**.
  Ej: **id_tabla INT NOT NULL PRIMARY KEY**.

    - Sus valores son únicos para cada registro y nunca nulos (NOT NULL).
    - Identifica a cada registro de la tabla.

- **Check:** especifica qué datos son aceptados en una columna. Nos permite definir el rango de valores que podremos introducir en una columna.
  Ej: **columnaTabla INT CHECK (columnaTabla > 700)**.

- **Unique:** sirve para determinar que todos los valores en la columna son únicos.
  Ej: **columnaTabla VARCHAR (255) UNIQUE**.

- **References (Foreign Key):** permite asociar una clave foránea de la tabla actual con la clave principal de otra tabla.
  Ej: **columnaTabla1 INT FOREIGN KEY REFERENCES tabla2(id_tabla2)**.

    - La clave foránea siempre tendrá un valor asociado con la tabla a la que hace referencia, pues la primary key de esta siempre tendrá valor **(integridad referencial)**.
    - Los campos que forman la clave foránea y principal deben ser del mismo tipo y tamaño.

Podemos determinar el comportamiento de la integridad referencial mediante las cláusulas:

- **On Delete/Update Cascade:** cuando se intenta borrar/actualizar un registro de la tabla padre **(la que tiene la primary key)** si este tiene registros asociados en la tabla hija **(la que tiene la foreign key)** el sistema los borra también.

- **On Delete/Update Restrict:** en este caso si la tabla padre tiene registros asociados en la tabla hija el sistema no permite realizar la **eliminación/actualización**.

DML. Operaciones sobre tablas en SQL.

Las operaciones básicas conocidas como **CRUD (Create, Read, Update y Delete)** que ofrece **SQL** son:

- **Insert:** añade filas a una tabla (puede insertar columnas específicas o varios registros a la vez).
  Ej: **INSERT INTO nuevaTabla VALUES (8, "cadena 1", "2001-08-10");**

- **Update:** actualiza los valores de campos y registros de una tabla (si no utilizamos la cláusula **WHERE** la modificación afectara a **todos** los registros de la tabla).
  Ej: **UPDATE nuevaTabla SET entero = 5, fecha = "1758-05-25" WHERE cadena = "cadena 5";**

- **Delete:** borra filas de una tabla (si no utilizamos la cláusula **WHERE** eliminaremos **todos** los registros de la tabla).
  Ej: **DELETE FROM nuevaTabla WHERE entero = 3;**

DELETE FROM nombreTabla sin la cláusula **WHERE** tiene como resultado una tabla **sin datos**, al igual que sucede al utilizar la sentencia **TRUNCATE TABLE nombreTabla**, sin embargo, esta última es **más rápida y eficaz**.

- **Select:** permite consultar registros de la base de datos.

    - Muestra las columnas indicadas de la tabla referenciada.
      Ej: **SELECT cadena, entero FROM nuevaTabla;**

    - Muestra todas las columnas de la tabla referenciada.
      Ej: **SELECT * FROM nuevaTabla;**

Podemos asignar un alias a las columnas que nos devuelve una consulta de la siguiente manera:

- **SELECT entero numero, cadena frase FROM nuevaTabla;**

Para evitar valores duplicados utilizaremos la cláusula DISTINCT:

- **SELECT DISTINCT entero FROM nuevaTabla;**

Si ejecutamos la sentencia **SELECT DISTINCT entero, cadena FROM nuevatabla;** el resultado serán **todos** los registros, pues tienen diferentes valores en alguna de las dos columnas

Se pueden añadir condiciones para devolver resultados filtrados:

- **SELECT * FROM nuevaTabla WHERE entero = 8 OR cadena = "cadena 2";**

Devuelve **todas** las columnas de la table referenciada que cumplan la condición.

Operadores de comparación:

- Utilizaremos los operadores lógicos **AND**, **OR** y **NOT** para concatenar condiciones.

    - **BETWEEN … AND … :** para indicar que un campo se encuentra entre dos valores numéricos o fechas.
      Ej: **SELECT * FROM nuevaTabla WHERE entero BETWEEN 5 AND 8;**

    - **IN:** puede trabajar con campos numéricos, texto y fecha. Su funcionamiento es como un = pero con una lista indeterminada de valores.
      Ej: **SELECT * FROM nuevaTabla WHERE cadena IN ("cadena 7", "cadena 2", "cadena 5");**

    - **LIKE:** nos permite filtrar campos de tipo texto utilizando 2 símbolos:

        - **%:** representa varios caracteres.
          Ej: **SELECT * FROM nuevaTabla WHERE cadena LIKE "%Dis";**
        - **_:** representa un carácter.
          Ej: **SELECT * FROM nuevaTabla WHERE cadena LIKE "cadena_";**

- **Ej: LIKE 'B_R%'**: empieza en **B**, **1 carácter**, una **R** y varios **caracteres** después.

    - **IS NULL:** permite comprobar si el contenido de un campo no tiene valor o el valor es desconocido.
      Ej: **SELECT * FROM nuevaTabla WHERE cadena IS NULL;**

Operadores lógicos:

- **AND:** La salida es cierta solo si ambas entradas son ciertas.
- **OR:** La salida es falsa solo si ambas entradas son falsas.
- **NOT:** La salida niega la entrada.

Utilizamos paréntesis para que no haya **errores** debido a las reglas de precedencia **(NOT → AND → OR)**.

Operadores aritméticos: permite modificar el valor de las columnas devueltas en la selección. El valor no cambia en la base de datos, solo en la selección devuelta.

- Ej: **SELECT entero*2, cadena FROM nuevaTabla;**

Operador de concatenación (CONCAT): nos permite unir dos o más campos de la tabla.

- Ej: **SELECT CONCAT (entero, "&", cadena), fecha FROM nuevaTabla;**

Clausula ORDER BY: permite determinar el orden del resultado de una consulta. Siempre será la última cláusula de un **SELECT**.

La ordenación puede ser ascendente **(ASC)** o descendente **(DESC)**.

- Ej: **SELECT * FROM resultados ORDER BY Resultado DESC;**

- **ASC** en caso de letras es de la **"a"** a la **"z"** y **DESC** es de la **"z"** a la **"a"**.

En caso de que se establezca la ordenación por dos campos, se ordenara en orden de declaración.

- Ej: **SELECT * FROM resultados ORDER BY Operacion, Resultado ASC;**

Consultas sobre varias tablas:

INNER JOIN: sólo mostrará las filas que tengan elementos asociados. Si hay filas en la tabla padre que no tienen elementos asociados en la tabla hija, no las mostrará.

- Devuelve solo las filas que tienen coincidencias en ambas tablas, según una condición de unión especificada.

    - **INNER JOIN:** sería seleccionar solo las filas donde hay un estudiante inscrito en un curso.
      Ej: **SELECT estudiantes.id, estudiantes.nombre, cursos.nombre_curso FROM estudiantes INNER JOIN cursos ON estudiantes.id = cursos.id_estudiante;**

LEFT JOIN: Devuelve todas las filas de la tabla izquierda (la primera mencionada) y las filas coincidentes de la tabla derecha (la segunda mencionada). Si no hay coincidencias en la tabla derecha, se llenan con **NULLs**.

- Devuelve todas las filas de la izquierda y valores de la derecha o **null**, según una condición de unión especificada.

    - **LEFT JOIN:** dará la lista completa de estudiantes y, si están inscritos, mostrará información sobre los cursos en los que están inscrito.
      Ej: **SELECT estudiantes.id, estudiantes.nombre, cursos.nombre_curso FROM estudiantes LEFT JOIN cursos ON estudiantes.id = cursos.id_estudiante;**

RIGHT JOIN: devuelve todas las filas de la tabla derecha **(la segunda mencionada)** y las filas coincidentes de la tabla izquierda **(la primera mencionada)**. Si no hay coincidencias en la tabla izquierda, se llenan con **NULLs**.

- Devuelve todas las filas de la derecha y valores de la izquierda o **null**, según una condición de unión especificada.

    - **RIGHT JOIN:** mostrará todos los cursos y, si hay estudiantes inscritos, mostrará información sobre esos estudiantes.
      Ej: **SELECT estudiantes.id, estudiantes.nombre, cursos.nombre_curso FROM estudiantes RIGHT JOIN cursos ON estudiantes.id = cursos.id_estudiante;**

FULL OUTER JOIN (UNION): devuelve todas las filas de ambas tablas. Si hay coincidencias, muestra la información correspondiente; si no hay coincidencias, llena con **NULLs** en las columnas de la tabla que no tiene coincidencias.

- Devuelve todas las filas de ambas tablas, rellenando con **NULLs** en caso de que no haya coincidencias.

MySQL no soporta FULL OUTER JOIN