SQL
(Structured Query Language - Lenguaje Estructurado de Consulta).

- Lenguaje de programación que se utiliza para administrar bases de datos relacionales.

Historia.

Estructura.

- Compuesto por comandos, cláusulas, operadores, funciones de agregado, procedimientos, disparadores, etc.

COMANDOS O INSTRUCCIONES.

- SQL está compuesto de varios sub-lenguajes, cada uno de ellos con su conjunto de comandos.
- Subconjuntos de instrucciones:
  - **DDL - LDD (Data Definition Language - Lenguaje de Definición de Datos).**
    - Permite crear y modificar las estructuras y objetos de las BD.
    - Comandos:
      - **CREATE**.
        - Para crear objetos.
      - **ALTER**.
        - Modificar los objetos.
      - **DROP**.
        - Eliminar objetos.
  - **DML - LMD (Data Manipulation Language - Lenguaje de Manipulación de Datos).**
    - Comandos o instrucciones para manipular los datos contenidos en las bases de datos.
    - Comandos:
      - **SELECT**.
        - Instrucción para realizar consultas.
      - **DELETE.**
        - Eliminación de datos.
      - **INSERT**.
        - Inserción de datos.
      - **UPDATE**.
        - Actualización o modificación de datos.
  - **DCL - LCD (Data Control Language - Lenguaje de Control de Datos).**
    - Incluye instrucciones para manejar transacciones y otorgar o denegar o permisos y privilegios.
    - Permite controlar el acceso a objetos y a datos.
    - Comandos:
      - **GRANT**.
        - Otorgar Permisos o privilegios.

- **REVOKE.**
  - Denegar Permisos o privilegios.
- **COMMIT**.
  - Confirmar transacción.
- **ROLLBACK.**
  - Deshacer transacción.

CLÁUSULAS.

- Añaden condiciones de modificación para los comandos estándar.
- Algunas:
  - **FROM**.
    - Especifica las tablas que contienen los registros seleccionar.
  - **WHERE**.
    - Especifica condiciones a cumplir por los registros a seleccionar.
  - **ORDER BY**.
    - Ordenar registros según un criterio.
  - **GROUP BY**.
    - Agrupar registros.
  - **HAVING**.
    - Permite crear condiciones para grupos.

SCRIPTS DE BASES DE DATOS.

- **Concepto**:
  - Secuencia de instrucciones de SQL almacenadas en un archivo de texto plano.
  - Dichas instrucciones indican al SGBD qué tiene que hacer y en qué orden debe hacerlo.
  - Para crear un script se puede emplear un editor de textos (bloc de notas de Windows), editores de código (Notepad++, Visual Studio Code), el editor SQL que incorporan los SGBD o programas exclusivos para crear código SQL (SQL Script Generator).
  - El archivo que contiene el script se guarda con la extensión ".sql".
- **Contenido**:
  - Pueden contener cualquier instrucción que permita crear base de datos, tablas, restricciones, índices, vistas, consultas, modificaciones, inserciones o eliminaciones de datos, etc.
  - También puede contener el código de programación creado con lenguajes de programación como Transact-SQL de SQL Server (T-SQL), PL/SQL de Oracle Database o SQL/PSM usado por la mayoría de los SGBDR (IBM DB2, MySQL, PostgreSQL, etc.).
  - El uso de estos lenguajes permite la creación de funciones, procedimientos almacenados y triggers o disparadores.
- **Utilización**:
  - Documentar las acciones que se están realizando para que sirvan de ayuda para el autor u otro usuario a la hora de conocer lo que se está realizando.
  - Documentar los cambios que se realizan en la Base de Datos, para ello se pueden incluir comentarios dentro de los scripts que incluyan fecha, autor, motivo del cambio, etc.
  - Copiar una Base de Datos total o parcialmente de un equipo a otro.

  - Consultar la estructura de la Base de datos y confirmar si está todo correcto o no.
  - Tener un Backup o copia de seguridad de la base de datos manteniendo una copia permanente de los pasos usados para crear y rellenar las bases de datos en el servidor.
  - Formar rápidamente a los nuevos empleados al permitirles encontrar problemas en el código, entenderlo o cambiarlo.
  - Realizar tareas administrativas como distribución, replicación, almacenamiento o sincronización de las bases de datos.
- **Ejecución de un script SQL en phpMyAdmin.**
  - Opciones**:**
    - Pulsar en el botón "Continuar".
    - Control + Enter.
- **PROCEDIMIENTO ALMACENADO**
  - **Concepto**:
    - Código o programa almacenado físicamente en la base de datos.
    - Puede contener cualquier tipo y cantidad de instrucciones SQL.
  - **Utilización**:
    - Crear objetos, realizar consultas, validar datos, mostrar información, controlar la gestión de operaciones, realizar procesos complejos, etc...
    - Se crean con la instrucción CREATE PROCEDURE.
- **TRIGGER, DESENCADENADOR O DISPARADOR**
  - **Concepto**:
    - Tipo de procedimiento almacenado que se ejecutan automáticamente cuando se produce una acción o evento en el servidor o en una base de datos.
    - Se ejecutan cuando el usuario intenta modificar datos de una tabla o vista.
    - Los eventos que los ejecutan por tanto son instrucciones INSERT, DELETE y UPDATE.
    - Se crean con la instrucción CREATE TRIGGER.
- **FUNCIONES**
  - **Concepto**:
    - Se usan para realizar operaciones o cálculos en general que pueden ser devueltos a una aplicación solicitante o integrados en un conjunto de resultados.
    - Tipos:
      - **Integradas**:
        - Funciones incluidas con MySQL para realizar múltiples operaciones.
        - Se agrupan en muchas categorías: cadena, numéricas, fecha y hora, agregada y otras.
      - **Usuario**:
        - Son las creadas por el usuario con CREATE FUNCTION.

ARCHIVOS DE BASES DE DATOS DE MySQL y MariaBD.

Motores de almacenamiento. (Storage-Engine).

- Un motor de almacenamiento es el encargado de almacenar, gestionar y recuperar toda la información de una tabla.
- A las tablas se les asigna un motor de almacenamiento determinado cuando son creadas.
- Se pueden guardar tablas en una misma base de datos que usen motores distintos.
- Ver los motores soportados MySQL o MariaDB:

- *show engines*;

- Especificar un motor de almacenamiento.
  - Al crear una tabla usar la instrucción engine.
  - Sintaxis:
    - engine = “Nombre del motor de almacenamiento”;
  - Ejemplo:
    - engine = “MyISAM”;

- **Tipos:**
  - **InnoDB**
    - Motor de almacenamiento de propósito general de MySQL.
    - Soporta transacciones ACID, integridad referencial y bloqueo de registros que permiten el acceso concurrente.
    - Útil en bases de datos complejas.
    - Mayor rendimiento con inserciones y actualizaciones.
  - **MyISAM**
    - Motor de almacenamiento original de MySQL.
    - **ISAM** (Indexed Sequential Access Method).
    - Método de acceso secuencial indexado.
    - El almacenamiento secuencial solo permite acceso para escribir por una aplicación a la vez, por lo que varios usuarios o aplicaciones no pueden escribir a la vez en la misma tabla.
    - No admite transacciones.
    - Si permite la lectura simultánea.
    - Utiliza 2 archivos para guardar los datos de una tabla:
      - Uno para guardar los datos en sí.
      - Otro para guardar el índice.
    - Útil en bases de datos pequeñas.
    - Más recomendable cuando se hacen más consultas que inserciones y modificaciones de datos.
  - **Aria**.
    - Alternativa a MyISAM tolerante a fallos, es decir, pretende ser una alternativa segura a MyISAM.
    - Válido para MySQL y MariaDB.
    - El objetivo es implementar todas las características del motor InnoDB.
  - **CSV (Comma-Separated Values).**
    - Almacena datos en archivos CSV.
    - Para diferenciar las columnas de la tabla se usan comas como separador, mientras que, un salto de línea representa a una fila de datos nueva.
  - **MRG_MyISAM**.
    - Tablas de combinación MyISAM.
    - Agrupa lógicamente de tablas MyISAM idénticas considerándolas un objeto único.

Extensiones y formatos de archivo en MySQL y MariaDB:

- **frm.**
  - Guarda el esquema o definición de las tablas y objetos contenidos en la BD,
  - Usado con tablas MyISAM e InnoDB.

- **opt.**
  - Archivo que almacena las características de la BD en el proceso de su creación.
  - Cada vez que se crea o modifica una base de datos, las características de ésta se almacenan en un archivo de texto con el nombre db.opt.
- **myd.**
  - Guarda los datos de tablas MyISAM.
- **myi.**
  - Archivo de índices de una tabla MyISAM.
- **idb (ibdata).**
  - Contiene los datos de las tablas InnoDB.
- **ibl (iblog).**
  - Contiene las transacciones.
  - Archivo de registro o log de las tablas InnoDB.
- **mad**.
  - Guarda los datos de tablas Aria.
- **mai**.
  - Archivo de índices de una tabla Aria.

DICCIONARIO – CATÁLOGO DE DATOS EN MYSQL.

- La mayor parte de la información sobre las bases de datos en MySQL se guarda en la base INFORMATION_SCHEMA.
- Consulta de información:
  - **Use information_schema**;
    - Pone en uso la base del sistema information_schema para su consulta.
  - **desc / describe schemata**;
    - Muestra los campos de la tabla schemata que contienen todas las bases de datos del servidor MySQL, incluidas las del sistema.
  - **Select * from schemata**;
    - Muestra las bases de datos contenidas en la tabla schemata.
  - **desc / describe tables**;
    - Muestra los campos contenidos en la tabla schemata.
  - **desc / describe columns**;
    - Muestra los campos contenidos en la tabla schemata.

UTILIZACIÓN DE MYSQL – MARIADB.

phpMyAdmin.

- Aplicación web que permite administrar bases de datos MariaDB - MySQL.
- Incorpora una interfaz gráfica, si bien permite también escribir scripts SQL para la gestión de una base de datos.
- Se basa en PHP.
- Uso desde la aplicación Xampp.

    - Servidores Web y MySQL deben estar ejecutándose.
    - Abrir una ventana en un navegador.
    - Conectarse al servidor local:
        - https://localhost
        - https://127.0.0.1
        - https://[::1]
- Dentro del panel de control o dashboard de Xampp abrir phpMyAdmin.
- Otra forma es desde el panel de control de Xampp pulsando en Admin para MySQL.

Modo consola.

- Abrir consola en Windows o de Powershell:
    - Ejecutar/CMD
    - Buscar CMD o Powershell y abrirlos.
    - Inicio/Programas/Sistema de Windows.
    - Inicio/Programas/Accesorios de Windows/Windows Powershell
- Navegar hasta la carpeta bin de MySQL o de MariaDB:
    - cd c:\xampp\mysql\bin <--- MariaDB / MySQL con Xampp.
    - cd c:\program files\mysql\mysql server 8.0\bin <--- MySQL con Workbench.
- Conectarse a la instancia de MySQL usando un nombre de usuario y una contraseña:
    - mysql -u “nombre de usuario” -p
    - Escribir contraseña. (Si no hay contraseña definida “Intro” sin escribir).
- Ejemplo:
    - mysql -u root –p
    - password: 1234.
- Salir de MySQL:
    - exit

COMANDOS DE MSDOS O WINDOWS.

- **cd.**
    - Permite abrir un directorio o carpeta, o cerrarlo y subir a un nivel superior.
    - Sintaxis:
        - cd nombre de directorio (Abre una carpeta)
        - cd ruta de acceso/nombre de directorio (Abre una carpeta especificando la ruta para llegar a ella).
        - cd.. (Cierra la carpeta o directorio actual).
    - Ejemplos:
        - cd Windows
        - cd C:\Windows\System32
        - cd..
- **dir.**
    - Muestra el contenido de un directorio.
    - Si la carpeta tiene mucho contenido y no se ve todo, se pueden hacer pausas usando el parámetro /p:

        - Ejemplo:
            - C:\Windows\System32>dir /p
    - **md.**
        - Crea una carpeta en la ubicación local.
        - Primero hay que ubicarse en la carpeta donde se va a crear la nueva usando el comando cd.
        - Sintaxis:
            - md nombre de la carpeta nueva.
        - Ejemplo:
            - md nueva
    - **rd**.
        - Borra una carpeta en la ubicación local.
        - Primero hay que ubicarse en la carpeta donde está la carpeta a borrar usando el comando cd.
        - Sintaxis:
            - rd nombre de la carpeta a borrar.
        - Ejemplo:
            - rd nueva
    - **del.**
        - Permite borrar un archivo local.
        - Hay que estar en la carpeta que contiene el archivo a borrar, para ello se usa cd.
        - Se pueden usar comodines (* y ?).
        - Sintaxis:
            - del nombre del archivo a borrar.
        - Ejemplo uso *:
            - del uno.html
            - del *.jpg
            - del pagina.*
            - del *.*
        - Ejemplo uso ?:
            - Hay que usar tantas interrogaciones como caracteres lleve el patrón.
            - L????.jpg borraría Luisa.jpg, Lucia.jpg, Lucas.jpg, pero no Luis.jpg que tiene un carácter menos que el patrón.
    - **exit.**
        - Cierra CMD o Powershell.

SINTAXIS SQL BÁSICA.

- Todas las instrucciones o sentencias acababan en punto y coma (;).
- **Comentarios**.
    - De una línea:
        - - - Texto del comentario.
    - De varias líneas:

        /* Texto del comentario

Texto del comentario
Texto del comentario. */

LENGUAJE LDD – DDL.

- Permite tanto la creación de las bases de datos, como de las estructuras y objetos que éstas pueden contener.
- Comandos: CREATE, ALTER y DROP.

CREACIÓN DE BASES DE DATOS.

- Para crear una base de datos hay 2 instrucciones diferentes que dan el mismo resultado.
- Sintaxis:
  - Básica:
    - CREATE DATABASE Nombre de la Base de Datos;
    - CREATE SCHEMA Nombre de la Base de Datos;
  - Completa:
    - CREATE DATABASE | SCHEMA [IF NOT EXISTS] Nombre de la Base de Datos
      [CHARACTER SET [=] Juego de caracteres por defecto.]
      [COLLATE [=] Modo de intercalación por defecto.]
      [ENCRYPTION [=] "Y" | "N"]

      Opciones:
    - **IF NOT EXISTS**.
      - Permite asegurarse de que la base de datos que se va crear no existe.
    - **CHARACTER SET**.
      - Juego de caracteres a utilizar.
    - **COLLATE**.
      - Intercalación o cotejamiento.
      - MySQL incluye juegos de caracteres que permiten almacenar datos usando una variedad de juegos de caracteres y realizar comparaciones de acuerdo con una variedad de intercalaciones.
      - Incluyen por ejemplo características sobre cómo se van a tratar las palabras acentuadas o no, las palabras en mayúsculas o minúsculas, etc.
      - Intercalación predeterminada para las bases de datos:
        - utf8mb4_general_ci.
    - **ENCRYPTION**.
      - Define el cifrado de base de datos predeterminado, que heredan las tablas creadas en la base de datos.
      - Por defecto, desactivado.
- Ejemplos:
  - Básica:
    - create database Banco;
    - create schema Tienda;

- Completa:

```
CREATE DATABASE IF NOT EXISTS Empresa
CHARACTER SET utf8
COLLATE utf8_general_ci;
```

ORDEN DE INTERCALACIÓN.

- El orden de intercalación especifica cómo se ordenarán los datos.
- Afecta a las operaciones de búsqueda porque especifica si se va a hacer distinción entre minúsculas y mayúsculas, palabras acentuadas o no, etc.
- Tipos de intercalación:
  - **_BIN**.
    - Ordenación binaria distinguiendo todos los caracteres.
  - **_CI_AI**.
    - No distingue mayúsculas y minúsculas, ni palabras acentuadas o no.
  - **_CI_AS**.
    - No distingue mayúsculas y minúsculas. Distingue entre palabras acentuadas o no.
  - **_CS_SI.**
    - Distingue entre mayúsculas y minúsculas. No distingue palabras acentuadas o no.
  - **_CS_AS**.
    - Distingue entre mayúsculas y minúsculas. Distingue entre palabras acentuadas o no.

Mostrar las bases de datos almacenadas.

- Sintaxis:
  - show databases;
  - show schemas;

Seleccionar la base de datos a usar.

- Para poder utilizar una base de datos hay que ponerla en uso.
- De esta forma MySQL sabe sobre qué base ejecutar los comandos SQL.
- Sintaxis:
  - use nombre de la base de datos;
- Ejemplo:
  - use Tienda;

MODIFICACIÓN DE BASES DE DATOS.

- Para crear una base de datos hay 2 instrucciones diferentes que dan el mismo resultado.
- Sintaxis:
  - ALTER DATABASE | SCHEMA Nombre de la Base de Datos
    [CHARACTER SET [=] Juego de caracteres por defecto.]

[COLLATE [=] Modo de intercalación por defecto.]
[ENCRYPTION [=] “Y” | “N”]

- Opciones:
  - **CHARACTER SET**.
    - Juego de caracteres a utilizar.
  - **COLLATE**.
    - Intercalación o cotejamiento.
    - MySQL incluye juegos de caracteres que permiten almacenar datos usando una variedad de juegos de caracteres y realizar comparaciones de acuerdo con una variedad de intercalaciones.
    - Incluyen por ejemplo características sobre cómo se van a tratar las palabras acentuadas o no, las palabras en mayúsculas o minúsculas, etc.
    - Intercalación predeterminada para las bases de datos:
      - utf8mb4_general_ci.
  - **ENCRYPTION**.
    - Define el cifrado de base de datos predeterminado, que heredan las tablas creadas en la base de datos.
    - Por defecto, desactivado.

- Ejemplos:
  - alter database Banco character set latin1;
  - alter schema Tienda collate utf8_unicode_ci encryption ”y”;

CAMBIAR EL NOMBRE DE UNA BASE DE DATOS.

- Cambiar el nombre de una base de datos implica borrar la base de datos y volverla a crear con un nuevo nombre.
- Para no perder las tablas, se cambia el nombre (opcional), y la ubicación de éstas usando RENAME.
- Sintaxis:
  - CREATE DATABASE <Base de datos con nuevo nombre>;
    USE <Base de datos con nuevo nombre>;
    RENAME TABLE <Base de datos antigua.Tabla> TO <Base de datos nueva.Tabla >;
    DROP DATABASE <Base de datos antigua>;
- Ejemplo:
  - CREATE DATABASE Nueva;
    USE Nueva;
    RENAME TABLE Antigua.Datos TO Nueva.Datos;
    DROP DATABASE Antigua;

ELIMINACIÓN DE BASES DE DATOS.

- Eliminar una base de datos implica eliminar sus tablas y todo su contenido.
- Sintaxis:

- drop database | schema nombre de la base de datos a eliminar;
- drop database | schema if exists nombre de la base de datos a eliminar;
- Opciones:
  - **IF EXISTS.**
    - Permite asegurarse de que la base de datos a eliminar existe.

- Ejemplos:
  - drop schema Tienda;
  - drop database if exists Empresa;

TABLAS.

- Estructura contenedora de los datos de una base de datos.

CREACIÓN DE TABLAS.

- Para crear una tabla hay poner en uso la base de datos donde se creará utilizando la instrucción USE.
- Sintaxis**:**

```
CREATE TABLE [IF NOT EXISTS] Nombre De La Tabla
(<Nombre Campo 1> <Tipo de dato> <Restricciones o Propiedades de Campo o Columna>,
<Nombre Campo 2> <Tipo de dato> <Restricciones o Propiedades de Campo o Columna>,
<Nombre Campo N> <Tipo de dato> <Restricciones o Propiedades de Campo o Columna>,
[RESTRICCIONES A NIVEL DE TABLA]);
Otras opciones para las tablas:
[ENGINE= Tipo de motor de almacenamiento]
[DEFAULT CHARSET=Juego de caracteres por defecto]
[COMMENT= “Comentario sobre la tabla”;]
```

- Ejemplos**:**

```
CREATE TABLE IF NOT EXISTS Persona (
Codigo varchar(5) PRIMARY KEY,
Nombre varchar (40) NOT NULL,
Apellido1 varchar (30) NOT NULL,
Apellido2 varchar (30) NOT NULL,
Dirección varchar(40) NOT NULL COMMENT “Dirección completa”,
Teléfono varchar (40) NOT NULL,
Edad int (3) NOT NULL,
Fecha_Nacimiento date NOT NULL,
PRIMARY KEY(Codigo)); //Alternativa a crear la restricción a nivel de columna. En este caso es a nivel de tabla.
```

TIPOS DE DATOS.

NUMÉRICOS ENTEROS.

- **INT (nº).**
  - Ocupa 4 bytes.

- El número entre paréntesis es opcional y representa la cantidad de dígitos que tendrá el número.
- Rango de valores:
  - -2.147.483.648 a 2.147.483.647.
  - 0 a 4.294.967.295.

- **SMALLINT (nº).**
  - Ocupa 2 bytes.
  - El número entre paréntesis es opcional y representa la cantidad de dígitos que tendrá el número.
  - Rango de valores:
    - -32.768 a 32.767.
    - 0 a 65.535.
- **TINYINT (nº).**
  - Ocupa 1 byte.
  - El número entre paréntesis es opcional y representa la cantidad de dígitos que tendrá el número.
  - Rango de valores:
    - -128 a 127.
    - 0 a 255.
- **MEDIUMINT (nº).**
  - Ocupa 3 bytes.
  - El número entre paréntesis es opcional y representa la cantidad de dígitos que tendrá el número.
  - Rango de valores:
    - -8.388.608 a 8.388.607
    - 0 a 16.777.215.
- **BIGINT (nº).**
  - Ocupa 8 bytes.
  - El número entre paréntesis es opcional y representa la cantidad de dígitos que tendrá el número.
  - Rango de valores:
    - - 9.223.372.036.854.775.807 a 9.223.372.036.854.775.807.
    - 0 a 18.446.744.073.709.551.615
- **BIT (Asimilable a BOOLEAN).**
  - Almacena los números enteros 0 o 1.
  - Pueden utilizarse respectivamente como un valor verdadero o un valor falso.

NUMÉRICOS REALES.

- **DECIMAL.**
  - Almacena valores numéricos exactos.
  - El tipo de datos NUMERIC se utiliza igual que el tipo DECIMAL en MySQL.
  - Sintaxis:
    - **Decimal (M, D):**
      - **M**:
        - Total de dígitos del número incluyendo la parte entera y la decimal (precisión).

        - Máximo 65.
    - **D:**
        - Número total de decimales (escala).
        - Máximo 30.
        - Si no se indica, el valor por defecto es 0.
  - Ejemplo:
    - decimal (6,3)
        - 456.912
        - 23.986

- **FLOAT.**
  - Ocupa 4 bytes.
  - Almacena valores numéricos aproximados, es decir, representa el valor más cercano al indicado.
  - Almacena números con decimales de precisión simple de hasta 7 decimales.
  - Sintaxis:
    - **Float**:
        - Representa un valor con coma flotante que no especifica un número de decimales.
    - **Float** (precisión).
        - Admite un valor entre 0 y 23 para un valor de precisión simple. Un valor entre 24 y 53 MySQL lo consideraría número como de tipo DOUBLE, por tanto, de doble precisión.
    - **Float** (M, D):
        - Sintaxis en desuso.
        - **M**:
            - Total de dígitos del número incluyendo la parte entera y la decimal.
        - **D:**
            - Número total de decimales.
        - Ejemplo:
            - float (8,2)
                - 985445,45
                - 445,45

- **DOUBLE.**
  - Ocupa 8 bytes.
  - Almacena valores numéricos aproximados.
  - Almacena números con decimales de precisión doble de hasta 15 decimales.
  - Sintaxis:
    - **Double**:
        - Representa un valor con coma flotante que no especifica un número de decimales.
    - **Double** (precisión).
        - Precisión: Valor entre 24 y 53.
    - **Double** (M, D):
        - Sintaxis en desuso.
        - **M**:
            - Total de dígitos del número incluyendo la parte entera y la decimal.
        - **D:**

        - Número total de decimales.
    - Ejemplo:
        - double (4,1)
            - 32.7
            - 872.4

FECHAS.

- **DATE.**
    - Almacena fechas con año, mes y día.
    - Rango de fechas:
        - “1000-01-01” y “9999-12-31”.
- **DATETIME.**
    - Almacena una fecha (año-mes-día) y hora (horas-minutos-segundos).
    - Rango de fechas y horas:
        - “1000-01-01 00:00:00” y “9999-12-31 23:59:59”.
- **TIME.**
    - Almacena una hora con hora, minutos y segundos.
    - Rango de horas:
        - 00:00:00 y 23:59:59.
    - Formato almacenado:
        - “HH:MM:SS”.
- **TIMESTAMP.**
    - Almacena una fecha y hora UTC.
    - Rango de fechas y horas:
        - “1970-01-01 00:00:01.999999” y “2038-01-19 03:14:07.999999”.
- **YEAR.**
    - Almacena un año determinado con 2 o 4 dígitos de longitud.
    - Por defecto 4 dígitos para el año.
    - Rango de años:
        - 1901 a 2155 con 4 dígitos.
        - 1970 a 2069 con 2 dígitos. (70-69).

CADENAS DE CARACTERES.

- **CHAR.**
    - Almacena cadenas de caracteres de tamaño fijo que, si es necesario, se rellenan por la derecha con espacios.
    - Longitud fija.
    - Rango de caracteres:
        - Entre 1 a 255.
- **VARCHAR.**
    - Almacena cadenas de caracteres de tamaño variable.
    - Rango de caracteres:
        - Entre 1 a 255.
- **TINYBLOB.**
    - BLOB**:**

        - Binary Large Object (Gran Objeto Binario).
    - Válido para objetos binarios como ficheros de texto, imágenes, audios o videos.
    - Longitud máxima:
        - 255 caracteres.
        - No distingue entre minúsculas y mayúsculas.
- **BLOB**
    - Igual que el anterior con una longitud máxima de 65.535 caracteres.
- **MEDIUMBLOB.**
    - Igual que el anterior con una longitud máxima de 16.777.215 caracteres.
- **LONGBLOB.**
    - Igual que el anterior con una longitud máxima de 4.294.967.298 caracteres.
- **TINYTEXT.**
    - Sirve para almacenar texto plano sin formato.
    - Longitud máxima:
        - 255 caracteres.
    - Distingue entre minúsculas y mayúsculas.
- **TEXT.**
    - Igual que el anterior con una longitud máxima de 65.535 caracteres.
- **MEDIUMTEXT**.
    - Igual que el anterior con una longitud máxima de 16.777.215 caracteres.
- **LONGTEXT.**
    - Igual que el anterior con una longitud máxima de 4.294.967.298 caracteres.
- **ENUM.**
    - Permite crear una lista de valores permitidos de los que se puede elegir uno.
    - La lista puede contener hasta 65.535 valores distintos.
    - El campo sólo puede tener uno de los valores permitidos en la lista.
    - Sintaxis:
        - Nombre del Campo ENUM (“valor 1”, “valor 2”, … , ”valor N”),
    - Ejemplo:
        - Provincia ENUM (“Toledo”, “Málaga”, “Sevilla”),
- **SET.**
    - Permite crear una lista de valores permitidos de los que se puede elegir uno.
    - La lista puede contener hasta 64 valores distintos.
    - El campo puede tener 0, 1 o varios de los valores incluidos en la lista.
    - Sintaxis:
        - Nombre del Campo SET (“valor 1”, “valor 2”, … , ”valor 64”),
    - Ejemplo:
        - Letras SET (“a”, “b”, “c”, “d”),

PROPIEDADES Y RESTRICCIONES.

NULL / NOT NULL.

- Establece si el valor del campo de incluirse obligatoriamente (NOT NULL) o no (NULL).
- Gestiona si una columna puede o no contener valores nulos.
- Si no se incluye, la opción por defecto es NULL, permitir valores nulos.
- Sintaxis:

  - Nombre de campo tipo de dato NULL | NOT NULL,
- Ejemplo:
  - autor varchar (50) NOT NULL,
  - email varchar (25) NULL,

DEFAULT.

- Propiedad que permite incluir valores por defecto en una columna.
- Dicho valor se incluirá automáticamente a una columna cuando no se especifique un valor determinado en ella al introducir los datos.
- Sintaxis:
  - Nombre de campo tipo de dato DEFAULT VALOR POR DEFECTO,
- Ejemplo:
  - autor varchar (40) NOT NULL DEFAULT “”, // Cadena vacía por defecto.
  - localidad varchar (30) DEFAULT “Madrid”,
  - poblacion int NULL DEFAULT 5000,

COMMENT.

- Permite añadir comentarios a las columnas o a la tabla.
- Puede servir para incluir información adicional sobre alguna característica especial del campo y puede usarse como documentación.
- Sintaxis:
  - Nombre de campo tipo de dato COMMENT “Comentario”,
- Ejemplo:
  - Código int PRIMARY KEY COMMENT “CLAVE PRINCIPAL”,

AUTO_INCREMENT.

- Permite crear incrementos automáticos para un campo.
- Al introducir datos en los registros, no se incluye datos en este tipo de campo ya se generan automáticamente.
- El campo debe ser de tipo entero.
- Por defecto, el valor inicial es 1, pero se puede especificar el número a partir del cual se iniciará el incremento.
- Pude usarse en campos clave principal.
- Sintaxis:
  - Nombre de campo tipo de dato AUTO_INCREMENT.
  - Nombre de campo tipo de dato AUTO_INCREMENT = VALOR INICIAL,
- Ejemplo:
  - Clave int NOT NULL AUTO_INCREMENT,
  - Clave int NOT NULL AUTO_INCREMENT = 50,

UNSIGNED.

- Impide que un campo numérico pueda contener valores negativos.
- Duplica el conjunto de valores posibles para el número enteros.
- Sintaxis:
    - Nombre de campo tipo de dato UNSIGNED.
- Ejemplo:
    - Edad int UNSIGNED NOT NULL,

ZEROFILL.

- Rellena con ceros hasta completar el ancho de visualización especificado en la columna.
- Si el ancho de visualización de una columna numérica es 10 y se escribe un número de 3 dígitos, las posiciones de los otros 7, hasta completar el ancho de 10, se rellenan con ceros.
- Ejemplo:
    - Valor introducido en el campo: 341.
    - Valor mostrado en el campo: 000000341.
- Su uso obliga a usar también UNSIGNED.
- Sintaxis:
    - Nombre de campo tipo de dato UNSIGNED ZEROFILL.
- Ejemplo:
    - Edad int(5) UNSIGNED ZEROFILL NOT NULL,

UNIQUE.

- Es una restricción que permite definir índices con claves únicas.
- La diferencia entre un índice único y uno normal es que el único no permite duplicados.
- Se pueden repetir nulos (NULL).
- Puede haber varios campos que tienen restricciones UNIQUE.
- Se puede utilizar con claves alternativas o secundarias.
- Sintaxis:
    - Nombre de campo tipo de dato UNIQUE, /// A nivel de columna.
    - UNIQUE (nombre del campo), //A nivel de tabla.
- Ejemplo:
    - DNI varchar(10) NOT NULL UNIQUE,
    - ISBN varchar(30) NOT NULL,
    - UNIQUE(ISBN),

PRIMARY KEY.

- Una clave primaria garantiza la **integridad de entidad:**
    - Restricción que identifica de forma única cada registro en una tabla.
- Una clave principal puede estar compuesta por uno o más campos.
- Un campo clave primaria no admite datos duplicados ni valores nulos.
- Solo puede haber una en una tabla.

- No se puede definir claves primarias múltiples a nivel de columna, sólo a nivel de tabla.
- Sintaxis:
    - Clave primaria simple:
        - Nombre de campo tipo de dato PRIMARY KEY, /// A nivel de columna.
        - PRIMARY KEY (nombre del campo), //A nivel de tabla con un solo campo.
        - CONSTRAINT nombre de restricción PRIMARY KEY (nombre del campo),
    - Clave primaria compuesta:
        - CONSTRAINT nombre de restricción PRIMARY KEY (campo 1, campo 2, …, campo N), //A nivel de tabla con varios campos.
- Ejemplos:
    - Clave primaria simple:
        - DNI varchar (10) PRIMARY KEY,
        - Codigo int (5) NOT NULL,
        - PRIMARY KEY (Codigo),
        - CONSTRAINT PK_codigo PRIMARY KEY (Codigo), // Dando nombre a la restricción.
    - Clave primaria compuesta a nivel de columna:
        - CodigoLibro varchar (10) PRIMARY KEY, // Incorrecto.
        - CodigoLector varchar (5) PRIMARY KEY, // Incorrecto.
    - Clave primaria compuesta a nivel de tabla:
        - CodigoLibro varchar (10),
        - CodigoLector varchar (5),
        - PRIMARY KEY (CodigoLibro, CodigoLector),
        - CONSTRAINT PK_biblio PRIMARY KEY (CodigoLibro, CodigoLector), // Dando nombre a la restricción.

FOREIGN KEY.

- Campo o campos que garantizan la integridad referencial entre tablas.
- **Integridad referencial.**
    - Asegura que se mantengan las referencias entre clave primaria y foránea.
    - De esta forma, no se puede eliminar ni modificar un registro si una clave ajena hace referencia a él.
- Una clave foránea o externa consiste en uno o varios campos en una tabla, que hacen referencia a la clave primaria de otra tabla.
- La tabla con la clave foránea se denomina tabla hija o secundaria, y la que tiene la clave principal con la que enlaza se denomina tabla padre, principal o de referencia.
- **Reglas:**
    - Los campos de clave foránea y primaria deben del mismo tipo, aunque no tienen por qué tener el mismo nombre.
    - Una clave foránea no puede modificarse, debe eliminarse y volverse a crear.
- Sintaxis:
    - Enlace a clave primaria simple:

        - FOREIGN KEY (Nombre del campo con clave foránea) REFERENCES <nombre de la tabla con clave primaria> (nombre del campo con clave primaria), //A nivel de tabla con un solo campo.
    - Enlace a clave primaria compuesta:
        - FOREIGN KEY (campo 1, campo 2, …, campo N) REFERENCES <nombre de la tabla con clave primaria> (campo 1, campo 2, …, campo N), //A nivel de tabla con varios campos.
- Ejemplos:
    - Enlace a clave primaria simple:
        - FOREIGN KEY (idsocio) REFERENCES Socios(idsocio);
    - Enlace a clave primaria compuesta:
        - FOREIGN KEY (IdDatos,idPro) REFERENCES Pedido (CodPedido, CodProducto);
- **Eliminación y actualización en cascada.**
    - Al existir integridad referencial entre tablas, al borrar un registro en la tabla padre se pueden eliminar todos los registros asociados en la tabla hija para evitar inconsistencias en las tablas.
    - Idéntica acción se puede realizar si se actualiza los registros o, todo lo contrario, se pueden bloquear estas acciones.
    - Opciones y Sintaxis**:**
        - **ON DELETE RESTRICT**
            - Acción predeterminada. No permite una eliminación si existe un registro asociado.
        - **ON DELETE NO ACTION**
            - No permite una eliminación si existe un registro asociado.
        - **ON DELETE CASCADE**
            - Eliminación en cascada.
        - **ON DELETE SET NULL**
            - No elimina los registros. Pone NULL en las claves foráneas de la tabla hija.
        - **ON UPDATE RESTRICT**
            - Acción predeterminada. No permite una eliminación si existe un registro asociado.
        - **ON UPDATE NO ACTION**
            - No permite una eliminación si existe un registro asociado.
        - **ON UPDATE CASCADE**
            - Actualización en cascada**.**
        - **ON UPDATE SET NULL**
            - Actualiza los registros asociados poniendo NULL en las calves foráneas de la tabla hija.
    - Ejemplos:
        - FOREIGN KEY (idsocio) REFERENCES Socios (idsocio) ON DELETE CASCADE ON UPDATE CASCADE;
        - FOREIGN KEY (CodProducto) REFERENCES Productos (CodProducto) ON UPDATE CASCADE;

CHECK.

- Sirve para restringir los valores permitidos en una columna o campo según cumplan o no éstos una condición.
- Formato de una condición:
  - (Campo operador valor).
- Operadores:
  - Para crear una condición se usan los siguientes operadores:
    - **Relacionales**:
      - =, <, >, <=, >=, <>
    - **Lógicos**:
      - AND, OR, NOT.
    - **Intervalos**:
      - BETWEEN… AND.
      - Sintaxis:
        - CHECK (BETWEEN Valor1 AND Valor 2),
      - Ejemplo:
        - CHECK (FECHA BETWEEN 100 AND 200),
        - CHECK (FECHA BETWEEN "1999-5-23" AND "2004-9-12"),
        - CHECK (FECHA >= "1999-5-23" AND FECHA <= "2004-9-12"), // Misma que anterior con operadores relacionales y lógicos.
    - **Patrones de valores**:
      - LIKE / NOT LIKE.
        - Comodines:
          - **%.**
            - Representa un conjunto de caracteres.
            - Se puede usar delante, detrás o en ambos lados del conjunto de caracteres fijos que tendrá el patrón.
            - Ejemplos:
              - %A - Todos los que terminen por "A".
              - ez% - Todos los que empiecen en "ez".
              - %mi% - Todos los que incluyen "mi" en la cadena.
          - **Guion de subrayado (_).**
            - Representa a un único carácter.
              - Ejemplo:
                - L_____ Representa a Lucia, Laura, etc.
          - **Corchetes [].**
            - Representa a uno de caracteres incluido dentro.
            - Se pueden usar rangos o caracteres sueltos.
            - Ejemplo:
              - "[abc]" – Que sea una de las siguientes letras a, b o c.
              - "[A-L]%" – Que empiece por un carácter comprendido entre A y L.
              - [0-9] [0-9] [0-9] – Tres números con cualquier dígito.

        - **Lista de valores**:
          - IN / NOT IN.
          - Sintaxis:
            - IN (Valor1, Valor 2, …, Valor N),

- Ejemplo:
  - IN (“Soltero”, “Casado”, “Separado”),
  - IN (234.00, 345.50, 450.23, 89.45, 673.17),

- Sintaxis:
  - Nombre de campo tipo de dato CHECK (condición), // A nivel de columna.
  - CHECK (condición), // A nivel de tabla. El campo al que afecta debe incluirse en la condición.
- Ejemplos:
  - A nivel de columna:
    - Edad int (3) UNSIGNED NOT NULL check (Edad >=18),
    - Provincia varchar(30) NOT NULL check IN (“Madrid”, “Toledo”),
  - A nivel de tabla:
    - Edad int (3) UNSIGNED NOT NULL,
    - Nombre varchar(50) NOT NULL,
    - Precio decimal(10,2) NOT NULL,
    - CHECK (Edad >=18);
    - CHECK (PRECIO BETWEEN 0.0 AND 100000.00),

MOSTRAR LAS TABLAS DE UNA BASE DE DATOS.

- Se ver cuáles son las tablas contenidas en una base de datos.
- La base de datos debe estar en uso.
- Sintaxis:
  - show tables;

CONSULTAR LA ESTRUCTURA DE UNA TABLA.

- Se puede mostrar la lista de campos con sus características (tipo de datos, campos clave, etc.).
- La base de datos que contiene la tabla a consultar debe estar en uso.
- Sintaxis:
  - describe nombre de la tabla;
  - desc nombre de la tabla
- Ejemplo:
  - describe alumnos;
  - desc alumnos;

MOSTRAR LAS RESTRICCIONES EXISTENTES EN UN BASE DE DATOS.

- Hay que realizar una consulta sobre la base de datos Information_Schema.
- Consulta:

```
SELECT CONSTRAINT_NAME, UNIQUE_CONSTRAINT_NAME, UPDATE_RULE, DELETE_RULE,
TABLE_NAME, REFERENCED_TABLE_NAME FROM
INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS WHERE CONSTRAINT_SCHEMA =
"Nombre de la base de datos";
```

MODIFICACIÓN DE TABLAS.

- Se pueden añadir, eliminar o modificar las columnas de una tabla.
- Se pueden añadir y eliminar las restricciones de una tabla.
- Se puede cambiar el nombre de la tabla o alguna de sus características (juego caracteres, motor de almacenamiento, etc.).
- **CAMBIAR EL NOMBRE A UNA TABLA.**
  - Sintaxis:
    - ALTER TABLE Nombre antiguo de la Tabla RENAME nombre nuevo de la Tabla.
  - Ejemplo:
    - ALTER TABLE Tabla1 RENAME Tabla2.
- **CAMBIAR EL JUEGO DE CARACTERES.**
  - Sintaxis:
    - ALTER TABLE Nombre de la Tabla CHARACTER SET Juego de caracteres.
  - Ejemplo:
    - ALTER TABLE Tabla1 CHARACTER SET latin1.
- **CAMBIAR MOTOR DE ALMACENAMIENTO.**
  - Sintaxis:
    - ALTER TABLE Nombre de la Tabla ENGINE Motor de almacenamiento.
  - Ejemplo:
    - ALTER TABLE Tabla1 ENGINE MyISAM.
- **AÑADIR UNA COLUMNA NUEVA AL FINAL DE LA TABLA.**
    - Sintaxis:
      - ALTER TABLE Nombre de la Tabla
        ADD <Nombre del Campo> <Tipo de Datos><propiedades>;
    - Ejemplo:
      - ALTER TABLE Tabla1
        ADD Codigo_Postal int (5) NOT NULL;
- **AÑADIR VARIAS COLUMNAS NUEVAS AL FINAL DE LA TABLA.**
    - Sintaxis:
      - ALTER TABLE Nombre de la Tabla
        ADD COLUMN <Nombre y Características del Campo Nuevo 1>, ADD COLUMN <Nombre y Características del Campo Nuevo 2>, ADD COLUMN <Nombre y Características del Campo Nuevo N>;
    - Ejemplo:
      - ALTER TABLE Tabla1
        ADD Codigo_Postal int (5) NOT NULL,
        ADD Provincia varchar (40) NOT NULL;

- **AÑADIR UNA COLUMNA NUEVA AL PRINCIPIO DE LA TABLA**.
  - También se pueden añadir varias combinándolas o no, con las otras opciones de inserción.
    - Sintaxis:
      - ALTER TABLE Nombre de la Tabla

ADD <Nombre del Campo> <Tipo de Datos> <propiedades> FIRST;

- Ejemplo:
  - ALTER TABLE Tabla1
    ADD Codigo_Postal int (5) FIRST;

- **AÑADIR VARIAS COLUMNAS NUEVAS AL PRINCIPIO DE LA TABLA.**
  - Sintaxis:
    - ALTER TABLE Nombre de la Tabla
      ADD COLUMN <Nombre y Características del Campo Nuevo 1> FIRST, ADD COLUMN <Nombre y Características del Campo Nuevo 2> FIRST, ADD COLUMN <Nombre y Características del Campo Nuevo N> FIRST;
  - Ejemplo:
    - ALTER TABLE Tabla1
      ADD Codigo_Postal int (5) NOT NULL FIRST,
      ADD Provincia varchar (40) NOT NULL FIRST;

- **AÑADIR UNA COLUMNA NUEVA A CONTINUACIÓN DE OTRA**.
  - También se pueden añadir varias combinándolas o no, con las otras opciones de inserción.
    - Sintaxis:
      - ALTER TABLE Nombre de la Tabla
        ADD <Nombre del Campo> <Tipo de Datos> <propiedades> AFTER <Nombre del Campo Anterior>;
    - Ejemplo:
      - ALTER TABLE Tabla1
        ADD Codigo_Postal int (5) DEFAULT 28000 AFTER Localidad;

- **AÑADIR VARIAS COLUMNAS NUEVAS A CONTINUACIÓN DE OTRAS.**
  - Sintaxis:
    - ALTER TABLE Nombre de la Tabla
      ADD COLUMN <Nombre y Características del Campo Nuevo 1> AFTER CAMPO, ADD COLUMN <Nombre y Características del Campo Nuevo 2> AFTER CAMPO, ADD COLUMN <Nombre y Características del Campo Nuevo N> AFTER CAMPO;
  - Ejemplo:
    - ALTER TABLE Tabla1
      ADD Codigo_Postal int (5) NOT NULL AFTER dni,
      ADD Provincia varchar (40) NOT NULL AFTER Codigo_Postal;

- **AÑADIR VARIAS COLUMNAS EN CUALQUIER POSICIÓN.**
  - Ejemplo:
    - ALTER TABLE Tabla1
      ADD Codigo_Postal int (5) NOT NULL AFTER dni, // Inserción antes del campo dni.
      ADD Provincia varchar (40) NOT NULL FIRST, // Insercion al principio.
      ADD Telefono varchar (10) NOT NULL; // Inserción al final.

- **CAMBIAR EL NOMBRE DE UNA COLUMNA**.

- Sintaxis:
  - ALTER TABLE Nombre de la Tabla
    CHANGE <Nombre del Campo Antiguo> <Nombre del Campo Nuevo><Tipo de Datos>;
- Ejemplo:
  - ALTER TABLE Tabla1
    CHANGE Codigo_Cliente CodCliente;

- **CAMBIAR EL TIPO DE DATO DE UNA COLUMNA**.
  - Sintaxis:
    - ALTER TABLE Nombre de la Tabla
      MODIFY <Nombre del Campo> <Tipo de Datos Nuevo>;
  - Ejemplo:
    - ALTER TABLE Tabla1
      MODIFY [COLUMN]CodCliente INT; (Con CHANGE también funciona):

- **CAMBIAR EL NOMBRE Y TIPO DE DATO DE UNA COLUMNA**.
  - Sintaxis:
    - ALTER TABLE Nombre de la Tabla
      CHANGE <Nombre del Campo Antiguo> <Nombre del Campo Nuevo> <Tipo de Datos Nuevo>;
  - Ejemplo:
    - ALTER TABLE Tabla1
      CHANGE Codigo_Cliente CodCliente INT NOT NULL;

- **ELIMINAR UNA COLUMNA**.
  - Sintaxis:
    - ALTER TABLE Nombre de la Tabla
      DROP COLUMN <Nombre del Campo a eliminar>;
  - Ejemplo:
    - ALTER TABLE Tabla1
      DROP COLUMN Titulo;
- **ELIMINAR VARIAS COLUMNAS**.
  - Sintaxis:
    - ALTER TABLE Nombre de la Tabla
      DROP COLUMN <Nombre del Campo a Eliminar 1>, DROP COLUMN <Nombre del Campo a Eliminar 2>, DROP COLUMN <Nombre del Campo a Eliminar N>;
  - Ejemplo:
    - ALTER TABLE Tabla1
      DROP COLUMN Precio, DROP COLUMN Saldo;
- **AÑADIR UN NUEVA RESTRICCION.**
  - Cada tipo de restricción tiene una sintaxis diferente.
    - Sintaxis genérica:
      - ALTER TABLE Nombre de la Tabla
        ADD CONSTRAINT <nombre de la restricción> <Tipo de restricción>;

- ALTER TABLE Nombre de la Tabla
  ADD <Tipo de restricción>;

- Ejemplos:
  - **Primary key**:
    - ALTER TABLE Tabla1
      ADD CONSTRAINT Primaria PRIMARY KEY (IdSocio);
    - ALTER TABLE Tabla1
      ADD PRIMARY KEY (IdSocio);
  - **Foreign key**:
    - ALTER TABLE Tabla2
      ADD CONSTRAINT Foránea FOREIGN KEY (CodSocio) references Tabla1(IdSocio);
    - ALTER TABLE Tabla2
    - ADD FOREIGN KEY (CodSocio) REFERENCES Tabla1(IdSocio);
  - **Unique**:
    - ALTER TABLE Tabla1
      ADD CONSTRAINT Unica UNIQUE (DNI);
    - ALTER TABLE Tabla1
    - ADD UNIQUE (DNI);
  - **Set Default**:
    - ALTER TABLE Tabla1
    - ALTER nombre SET DEFAULT "Pepe";
  - **Check**:
    - ALTER TABLE Tabla1
      ADD CONSTRAINT saldo_positivo CHECK (Saldo > 0.0);
    - ALTER TABLE Tabla1
      ADD CHECK (Saldo > 0.0);

- **BORRAR UNA RESTRICCION.**
  - Se utiliza el nombre de la restricción para borrar esta, si bien hay variaciones en la sintaxis según restricción.
  - Sintaxis:
    - ALTER TABLE Nombre de la Tabla
    - DROP <Tipo de restricción> <nombre de la restricción o del índice>;
    - ALTER TABLE Nombre de la Tabla
    - DROP CONSTRAINT <nombre de la restricción o del índice>;

  - Ejemplos:
    - **Primary key**:
      - ALTER TABLE Tabla1
        DROP PRIMARY KEY;
    - **Foreign key**:
      - ALTER TABLE Tabla2
        DROP FOREIGN KEY Foránea
    - **Unique**:
      - ALTER TABLE Tabla1

DROP INDEX Unica;

- **Default**:
  - ALTER TABLE Tabla1
  - ALTER Nombre DROP DEFAULT;
- **Check**:
  - ALTER TABLE Tabla1
    DROP CHECK saldo_positivo;
  - ALTER TABLE Tabla1
    DROP CONSTRAINT saldo_positivo;

- **MODIFICAR UNA RESTRICCION.**
  - Las restricciones no se pueden modificar, para cambiar una por otra hay que hacer lo siguiente:
    - Borrar la restricción a cambiar (ALTER TABLE ... DROP…).
    - Añadir la nueva restricción (ALTER TABLE...ADD…).

ELIMINACIÓN DE TABLAS.

- **DROP.**
  - Eliminar una tabla por completo, es decir, elimina su estructura y sus datos.
- **TRUNCATE.**
  - Elimina el contenido de la tabla, es decir, elimina los datos, pero no la estructura.
  - Sintaxis:
    - DROP TABLE [IF EXISTS] Nombre de la tabla;
    - TRUNCATE TABLE Nombre de la tabla;
    - Opciones:
      - **IF EXISTS.**
        - Permite asegurarse de que la tabla a eliminar existe.
    - Ejemplos:
      - DROP TABLE IF EXISTS Vendedores;
      - TRUNCATE TABLE Vendedores;

Reparación de una tabla.

- Si una tabla se corrompe o deja de funcionar correctamente MySQL puede repararla.
- **Chequear estado.**
  - Para conocer si una tabla tiene problemas y cuales son estos puede chequearse antes.
  - Sintaxis:
    - CHECK TABLE nombre de la tabla;
  - Ejemplo:
    - CHECK TABLE clientes; // Habría que poner en uso la base de datos que contenga la tabla clientes.
    - CHECK TABLE mysql.tables_priv; // Si se incluye el nombre de la base (mysql), no es necesario ponerla en uso.
- **Reparación**.

- Sintaxis:
  - REPAIR TABLE nombre de la tabla;
- Ejemplo:
  - REPAIR TABLE clientes; // Habría que poner en uso la base de datos que contenga la tabla clientes.
  - REPAIR TABLE suministros.clientes; // No necesario poner en uso la base suministros.
  - REPAIR TABLE mysql.tables_priv; // Si se incluye el nombre de la base (mysql), no es necesario ponerla en uso.

LENGUAJE LMD - DML.

OPERACIONES DE MODIFICIACIÓN DE DATOS.

- Para realizarlas se emplea el lenguaje de modificación de datos LMD o DML de SQL.
- Las operaciones a realizar son:
  - Consultas.
  - Inserción de registros nuevos.
  - Modificación de registros.
  - Eliminación de registros.
- Comandos: INSERT, UPDATE, DELETE y SELECT.

MODOS.

- **Modo Estricto.**
  - Función que gestiona como manejar los valores faltantes o no válidos en los campos cuando se insertan o actualizan registros.
  - Mostrar que modos están activos:
    - show variables like "sql_mode";
    - Los modos que interesa activar o desactivar para el modo estricto son:
      - STRICT_TRANS_TABLES.
      - STRICT_ALL_TABLES.
  - Desactivación temporal.
    - Uso de la instrucción set sql_mode = 'MODOS';
    - **Desactivar todos los modos**:
      - set sql_mode = ""; // Sin valores o instrucción vacía elimina todos los modos.
    - **Desactivar algún modo**:
      - set sql_mode = 'Modos que se quieren mantener y se quitan los que no'.
    - **Activar modos**:
      - set sql_mode = 'Modos a activar';
    - Ejemplos:
      - set sql_mode = "IGNORE_SPACE, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, NO_ENGINE_SUBSTITUTION"; // Al no estar incluidos los modos estrictos, éstos se desactivan, pero se mantiene los que se han especificado con *set sql_mode.*
  - Desactivación permanente.

        - Editar los archivos *my.cnf* o *my.ini* (según sea MySQL o MariaDB).
        - Buscar la entrada *sql_mode*
        - Eliminar de ella las opciones:
            - STRICT_TRANS_TABLES.
            - STRICT_ALL_TABLES.
        - Guardar cambios.
        - Si el archivo o la variable *sql_mode* no existen, se crean y se establece como vacía la variable en la sección[mysql].
- **Modo Seguro.**
    - Función que limita las acciones que se pueden realizar para evitar cometer errores que comprometan la integridad de la base de datos.
    - En concreto, para evitar errores al actualizar o eliminar registros cuando se hacen consultas sin usar el/los campo/s clave primaria en la cláusula *WHERE*, algunos SGBD tienen por defecto activado el modo seguro (por ejemplo, Workbench).
    - Para activar o desactivar dicho modo se usa la siguiente instrucción:
        - **Desactivar**:
            - SET SQL_SAFE_UPDATES = 0;
        - **Activar:**
            - SET SQL_SAFE_UPDATES = 1;

INSERT.

- Comando para insertar registros nuevos en una tabla.
- Si se quieren insertar varios registros con una misma instrucción insert, a continuación de los datos de un registro se pone una coma y se añaden los nuevos datos entre paréntesis, y así sucesivamente.
    - Sintaxis:
    - **Opción 1**.
        - INSERT INTO <Nombre de la Tabla> VALUES (Valor1, Valor2, ..., Valor N);
        - Esta opción permite incluir un valor por columna en el mismo orden, del mismo tipo y en toda la tabla.
        - Fechas y textos se escriben entre comillas.
        - Para que en un campo se incluya el valor por defecto, se escribe como dato: DEFAULT.
        - En un campo que admite nulos se puede incluir NULL.
    - **Opción 2**.
        - INSERT INTO <Nombre de la Tabla> (Campo 1, Campo 2, ..., Campo N) VALUES (Valor1, Valor2, ..., Valor N);
        - Con esta opción sólo se incluyen datos en los campos especificados.
        - También sirve para cambiar el orden de los campos y de la entrada de datos en éstos.
        - Si se omiten campos, por defecto se insertará en ellos el valor NULL.
        - Hay que tener cuidado con campos que incorporan la restricción NOT NULL, o que son claves.
        - Para un campo con tipo de datos que se incrementan automática y secuencialmente no se escribe ningún valor y no se incluye el campo en la lista de campos.

    - Ejemplos:
        - INSERT INTO Coches VALUES (“2345-WCX”, “Seat”, “Ibiza”, 12000.00); // Insertar un registro.

        - INSERT INTO Coches VALUES (“2345-WCX”, “Seat”, “Ibiza”, 12000.00), (“5478-ADR”, “Ford”, “Fiesta”, 10000.00); // Insertar varios registros.
        - INSERT INTO Coches VALUES (“2345-WCX”, “Seat”, “Ibiza”, 12000.00);
        - INSERT INTO Coches VALUES (“5478-ADR”, “Ford”, “Fiesta”, 10000.00); // Insertar varios registros de uno en uno. Equivalente a la sentencia anterior.
        - INSERT INTO Agenda VALUES (“Luis”, default); // El valor default incluirá el valor por defecto definido en el campo.
        - INSERT INTO Agenda (Nombre) VALUES (“Luis”); // El valor default incluirá el valor por defecto definido en el campo.

UPDATE.

- Permite modificar o actualizar el contenido de una tabla.
- Sintaxis:
    - **Modificar el contenido de un sólo campo**.
        - UPDATE <Nombre de la Tabla>
          SET <Nombre del Campo1> = Valor;
          [WHERE condición;]
    - **Modificar el contenido de varios campos**.
        - UPDATE <Nombre de la Tabla>
          SET <Nombre del Campo 1> = Valor,
          <Nombre del Campo 2> = Valor,
          <Nombre del Campo N> = Valor;
          [WHERE condición;]
          [LIMIT Número de registros];

        - LIMIT establece el límite máximo de registro a modificar.

DELETE.

- Comando de permite borrar registros de una tabla.
- TRUNCATE también elimina los registros de una tabla sin eliminar su estructura, pero se diferencia de DELETE en:
    - Borra más rápido que DELETE.
    - Reinicia un campo AUTO_INCREMENT y DELETE no.
    - No pide confirmación del borrado.
- Sintaxis:
    - DELETE FROM <Nombre de la tabla>;
        - Borra todos los registros de la tabla.
    - DELETE FROM <TABLA> WHERE Condición [LIMIT Número de registros];
        - Borra los registros que cumplan una condición determinada.
        - LIMIT establece el límite máximo de registro a borrar.
- Ejemplos:
    - DELETE FROM Coches;
    - DELETE FROM Coches WHERE Marca = “Seat”;

- DELETE FROM Coches WHERE Precio BETWEEN 10000.00 AND 20000.00;

SELECT.

- Comando que permite realizar consultas en una base de datos.
- El resultado de una consulta es un conjunto de registros que pueden cumplir ninguna, una o varias condiciones para ser mostrados.
- Sintaxis:
  - **Opción 1.**
    - Mostrar el contenido de un solo campo de una o más tablas:
      - SELECT <Nombre del Campo> FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...]; [WHERE Condición;]
  - **Opción 2.**
    - Mostrar el contenido de varios campos de una o más tablas:
      - SELECT <Nombre del Campo1, Nombre del Campo 2, …, Nombre del Campo N> FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...]; [WHERE Condición;]
  - **Opción 3.**
    - Mostrar el contenido de todos los campos de una o más tablas:
      - SELECT * FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...]; [WHERE Condición;]
- **Calificación de campos.**
  - Si dos o más campo de una o varias tablas tienen el mismo nombre es aconsejable calificarlos para evitar ambigüedades.
  - Para ello se especifica a qué tabla pertenece cada campo.
  - Si no hay ambigüedad con campos del mismo nombre entre tablas, se pueden poner diferentes campos en el orden que se quiera, independientemente del orden de las tablas que se incluyan.
  - Sintaxis:
    - Tabla.campo.
  - Ejemplo:
    - Personas.DNI.
    - Clientes.DNI.

AS.

- Cláusula que permite crear un alias.
  - **Para campos.**
    - Un alias es un nombre que hace más comprensible el resultado de una consulta cambiando los encabezados del resultado.
    - Aparecerá otro nombre en la columna como encabezado. Se usa
    - Sintaxis:
      - SELECT <Nombre del Campo 1>, AS “ALIAS” [, <Nombre del Campo 2> AS “ALIAS”, ... <Nombre del Campo N> AS “ALIAS”,] FROM <Nombre de la Tabla 1> [, <Nombre de la Tabla 2>, …];
    - Ejemplo:

        - SELECT NomCli AS “Nombre del Cliente” FROM Clientes;
    - **Para tablas.**
        - Permite simplificar el nombre de las tablas cuando se usan frecuentemente en las sentencias.
        - Sintaxis:
            - <Nombre de la Tabla 1> AS Alias [, <Nombre de la Tabla 2> AS Alias, …, <Nombre de la Tabla N> AS Alias …];
        - Ejemplo:
            - Suponiendo que hay un campo “Nombre” de dos tablas distintas “Morosos1” y “Morosos2”
            - SELECT m2.nombre, m1.nombre FROM morosos1 AS m1, morosos2 AS m2;

DISTINCT.

- Se usa cuando hay multiconjuntos en el resultado de la consulta, es decir, tuplas o registros repetidos.
- Muestra una única ocurrencia de cada registro repetido.
- Sintaxis:
    - SELECT DISTINCT <Nombre del Campo> FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
    - [WHERE Condición;]
    - SELECT DISTINCT <Nombre del Campo1, Nombre del Campo 2, …, Nombre del Campo N> FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
    - [WHERE Condición;]
    - SELECT DISTINCT * FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
    - [WHERE Condición;]
- Ejemplos:
    - SELECT DISTINCT Nombre FROM Clientes;

ALL.

- Muestra multiconjuntos, es decir, tuplas repetidas.
- Es la opción por defecto al usar SELECT, por lo que no es necesario incluirla.
- Sintaxis:
    - SELECT ALL <Nombre del Campo> FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
    - [WHERE Condición;]
    - SELECT ALL <Nombre del Campo1, Nombre del Campo 2, …, Nombre del Campo N> FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
    - [WHERE Condición;]
    - SELECT ALL * FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
    - [WHERE Condición;]
- Ejemplos:
    - SELECT ALL Nombre FROM Clientes;

WHERE.

- Permite establecer condiciones en una consulta, para así solo mostrar aquellos registros que la cumplan.
- Sintaxis:
    - WHERE condición.
    - Las condiciones pueden ser simples o compuestas e incluir diversos operadores.

- Formato de una condición:
  - WHERE Campo operador valor.
- Operadores:
  - Para crear una condición se usan los siguientes operadores:
    - **Relacionales**:
      - =, <, >, <=, >=, <>
    - **Lógicos**:
      - AND, OR, NOT.
    - **Intervalos**:
      - BETWEEN… AND.
        - Sintaxis:
          - Campo BETWEEN Valor1 AND Valor 2,
        - Ejemplo:
          - CANTIDAD BETWEEN 100 AND 200,
          - FECHA BETWEEN “1999-5-23” AND “2004-9-12”,
          - FECHA >= “1999-5-23” AND FECHA <= “2004-9-12”, // Misma que anterior con operadores relacionales y lógicos.
    - **Patrones de valores**:
      - LIKE / NOT LIKE.
        - Comodines:
          - **%.**
            - Representa un conjunto de caracteres.
            - Se puede usar delante, detrás o en ambos lados del conjunto de caracteres fijos que tendrá el patrón.
            - Ejemplos:
              - Nombre LIKE “A%” – Nombres que empiecen por “A”.
              - Apellidos LIKE “%ez” – Apellidos que terminen en “ez”.
              - Ciudades LIKE %mi% - Todas las ciudades que incluyen “mi” en la cadena.
              - Nombre NOT LIKE “A%” – Nombres que no empiecen por “A”.
          - **Guion de subrayado (_).**
            - Representa a un único carácter.
              - Ejemplo:
                - Nombre LIKE “L_____” // Mostraría Lucia, Laura, etc.
    - **Lista de valores**:
      - IN / NOT IN.
        - Sintaxis:
          - Campo IN (Valor1, Valor 2, …, Valor N),
        - Ejemplo:
          - Estado IN (“Soltero”, “Casado”, “Separado”),
          - Precio IN (234.00, 345.50, 450.23, 89.45, 673.17),
          - Color NOT IN (“Rosa”, “Verde”, “Amarillo”),
    - **Nulos / No nulos**.
      - Se pueden mostrar los registros que para algún campo tengan contenido o estén vacíos.

        - Por defecto, si no se incluye ninguna opción se muestran los registros con campos nulos y no nulos.
        - Sintaxis:
            - Campo IS NULL;
            - Campo IS NOT NULL;
        - Ejemplos:
            - Concepto IS NULL;
            - Portal IS NOT NULL;
    - **Expresión regular para rangos.**
        - REGEXP.
            - Permite mostrar los registros que coincidan con la expresión regular.
            - Caracteres para expresiones:
                - **^.**
                    - Lo que empiece por.
                - **$**
                    - Lo que termine por.
                - **[].**
                    - Lo que coincida con un carácter de los especificados en el rango o en el intervalo.
            - Sintaxis:
                - REGEXP Expresión;
            - Ejemplos:
                - Codigo regexp "[A-Z][1-5][2-5][S-Z]"; // F13Y Correcto, A63T Incorrecto.
                - Nombre regexp "^[ART]"; // Nombres que empiecen por A, R o T.
                - Nombre regexp "[ART]$"; // Nombres que terminen por A, R o T.
- Ejemplos Instrucción SELECT con WHERE.
    - SELECT Portal FROM Morosos WHERE Portal IN ('1','2'); // Muestra los portales 1 y 2.
    - SELECT * FROM Morosos WHERE Fecha >= "2013-02-23" AND "2013-09-12"; //Muestra todos los campos de aquellos registros incluidos en el intervalo temporal.
    - SELECT * FROM Morosos WHERE portal = "10"; // Muestra todos los morosos que viven en el portal 10.
    - SELECT * FROM Morosos WHERE portal IS NULL; // Muestra todos registros que tienen vacío el campo portal.

LIMIT.

- Indica el número de registros o filas que deben aparecer en el resultado de una consulta.
- Se incluye al final de la instrucción SELECT.
- Sintaxis:
    - LIMIT Número de registros a mostrar;
    - LIMIT Número como registro inicial a mostrar exclusive, número de registros a mostrar;
- Ejemplos:
    - SELECT * FROM Clientes LIMIT 10;
    - SELECT * FROM Clientes WHERE Saldo >= 20000.00 LIMIT 5;
    - SELECT * FROM Clientes LIMIT 5, 20; // Muestra 20 registros del resultado empezando por el sexto.

DUPLICAR UNA TABLA.

- **DENTRO DE LA MISMA BASE DE DATOS.**
  - Se puede hacer una copia de todo o parte del contenido de una tabla en otra nueva, o solo de la estructura de la tabla sin contenido.
  - Se pueden incluir algunos o todos campos de la tabla que se va a copiar.
  - Se pueden insertar los registros de una tabla en la copia vacía que sólo tiene la estructura.
  - Sintaxis:
    - **Sólo la estructura**.
      - CREATE TABLE <Nombre de la Tabla nueva>
        LIKE <Nombre de la Tabla a copiar>
    - **Estructura y datos.**
      - CREATE TABLE <Nombre de la Tabla nueva>
        SELECT <Campos Tabla a Copiar> FROM <Tabla a Copiar>
    - **Introducir los registros de la original en la copia vacía.**
      - INSERT INTO <Nombre de la Tabla nueva>
        SELECT <Campos Tabla a Copiar> FROM <Tabla Original>
      - Debe haber correspondencia exacta en cuanto a las columnas y tipos de datos.
  - Ejemplos:
    - CREATE TABLE MorososCopia1 LIKE Morosos;
    - CREATE TABLE MorososCopia2 SELECT * FROM Morosos;
    - CREATE TABLE MorososCopia3 SELECT Portal, Nombre, Concepto FROM Morosos;
    - INSERT INTO MorososCopia1 SELECT * FROM Morosos;

- **EN OTRA BASE DE DATOS.**
  - El proceso es el mismo que para hacerlo en la misma base, sólo que hay que calificar las tablas especificando las bases de datos de origen y destino.
  - Sintaxis:
    - **Sólo la estructura**.
      - CREATE TABLE <Nombre Base Datos Destino.Nombre de la Tabla nueva>
        LIKE < Nombre Base Datos Origen.Nombre de la Tabla a copiar>
    - **Estructura y datos.**
      - CREATE TABLE < Nombre Base Datos Destino.Nombre de la Tabla nueva>
        SELECT <Campos Tabla a Copiar> FROM < Nombre Base Datos Origen.Tabla a Copiar>
    - **Introducir los registros de la original en la copia vacía.**
      - INSERT INTO < Nombre Base Datos Destino.Nombre de la Tabla nueva>
        SELECT <Campos Tabla a Copiar> FROM < Nombre Base Datos Origen.Tabla Original>
      - Debe haber correspondencia exacta en cuanto a las columnas y tipos de datos.
  - Ejemplos:
    - CREATE DATABASE Morosos2;
    - CREATE TABLE Morosos2.MorososCopia1 LIKE Morosos.Morosos;
    - CREATE TABLE Morosos2.MorososCopia2 SELECT * FROM Morosos.Morosos;
    - CREATE TABLE Morosos2.MorososCopia3 SELECT Portal, Nombre, Concepto FROM Morosos.Morosos;
    - INSERT INTO Morosos2.MorososCopia1 SELECT * FROM Morosos.Morosos;

ORDER BY.

- Permite ordenar el resultado de una consulta.
- La ordenación puede ser ascendente (A-Z / 0-9) o descendente (Z-A / 9-0).
- Ordenación predeterminada: Ascendente.
- Hay que especificar uno o más campos como criterio de ordenación.
- Se pueden usar varios campos para ordenar cuando existen repeticiones en uno de ello que no quedan bien ordenadas, de esta forma un siguiente campo ordena lo repetido en el anterior.
- Cuando se usan varios campos como criterio de ordenación, se pueden ordenar todos de la misma o distinta forma, indicándoselo a cada campo.
- Sintaxis:
    - SELECT <Nombre del Campo> FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
      [WHERE Condición;]
      ORDER BY <Campo Criterio 1, Campo Criterio 2, …, Campo Criterio N> [ASC | DESC];
    - SELECT <Nombre del Campo1, Nombre del Campo 2, …, Nombre del Campo N>
      FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
      [WHERE Condición;]
      ORDER BY <Campo Criterio 1, Campo Criterio 2, …, Campo Criterio N> [ASC | DESC];
    - SELECT * FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
      [WHERE Condición;]
      ORDER BY <Campo Criterio 1, Campo Criterio 2, …, Campo Criterio N> [ASC | DESC];
    - SELECT * FROM Nombre de la Tabla1 [, Nombre de la Tabla2, ...];
      [WHERE Condición;] // Varios campos con el mismo tipo de ordenación.
      ORDER BY <Campo Criterio 1 [ASC | DESC], Campo Criterio 2 [ASC | DESC], …,
      Campo Criterio N [ASC | DESC]>; // Varios campos con distinto tipo de ordenación.

- **Uso con LIMIT**.
    - Si se combina con LIMIT se pueden obtener el número de los valores mayores o menores que se especifiquen.
    - LIMIT sería la última instrucción para incluir en la sentencia.
- Ejemplos:
    - SELECT * FROM Morosos ORDER BY Portal ASC;
    - SELECT Nombre, Fecha, cantidad FROM Morosos ORDER BY Cantidad DESC;
    - SELECT * FROM Morosos ORDER BY Cantidad DESC LIMIT 10; // Muestra las 10 cantidades más debidas.
    - SELECT * FROM Morosos ORDER BY Apellido1, Apellido2, Nombre ASC;
    - SELECT * FROM Morosos ORDER BY Apellido1 ASC, Apellido2 ASC, Nombre DESC;

FUNCIONES DE AGREGACIÓN, AGREGADAS O DE AGREGADO.

- Permiten realizar cálculos básicos para mostrar datos resumidos.
- Estas funciones se utilizan habitualmente junto con la cláusula GROUP BY.
- Si usan en una consulta sin agrupamiento de datos, agrupan todos los registros mostrando una sola fila como resultado.
- La mayoría deben utilizar con campos numéricos.
- Puede incluirse condiciones, por lo que los cálculos se hacen sólo sobre los registros que cumplan éstas.

- Se puede incluir un alias para la cabecera de los resultados.
- Se pueden incluir varias en una consulta.
- Funciones y sintaxis:
    - **MAX(**Campo**).**
        - Muestra el máximo valor de los incluidos en el campo.
    - **MIN(**Campo**).**
        - Muestra el mínimo valor de los incluidos en el campo.
    - **AVG(**Campo**).**
        - Realiza la media aritmética de todos los valores incluidos en el campo.
    - **SUM(**Campo**).**
        - Suma todos los valores incluidos en el campo.
    - **COUNT(**Campo**).**
        - Cuenta ocurrencias para el campo.
- Ejemplos**:**
    - SELECT MAX (cantidad) FROM Morosos**;**
    - SELECT MIN (cantidad) FROM Morosos WHERE fecha > ”2014-05-01”**;**
    - SELECT SUM (cantidad) as “Total debido en el 2014” FROM Morosos WHERE fecha >= ”2014-01-01” AND ”2014-12-31”**;**

FUNCIONES MATEMÁTICAS PREDETERMINADAS.

FORMAT.

- Aplica formatos numéricos que incluyen separador de millares, redondeo y número de decimales.
- Sintaxis:
    - FORMAT (Campo, Número de decimales, ”Código de Idioma”);
    - Código de idioma para español: “es_ES”.
- Ejemplo:
    - FORMAT (Cantidad, 2, “es_ES”); // Si cantidad tuviese el valor 23456.234 el resultado sería 23,456.23.
    - FORMAT (AVG (Cantidad), 0,“es_ES”); // Formatea el número resultante de la media de la cantidad.

ROUND.

- Redondea el valor del campo y muestra el número de decimales que se especifique.
- Sintaxis:
    - ROUND (Campo, Número de decimales);
- Ejemplo:
    - ROUND (Cantidad, 0); // Si cantidad tuviese el valor 45.89 el resultado sería 46.
    - ROUND (AVG(Cantidad), 0); // Redondea el número resultante de la media de la cantidad.

FUNCIÓN DE CONVERSIÓN.

CAST.

- Permite convertir un campo de un tipo a otro para un cálculo o para crear una expresión, pero no cambia el tipo en la estructura de la tabla.
- Sintaxis:

  - CAST (Campo AS Nuevo Tipo);
- Ejemplo:
  - CAST (Cantidad AS INT);
  - CAST (Portal AS INT);

CONVERT.

- Permite convertir un campo de un tipo a otro para un cálculo o para crear una expresión, pero no cambia el tipo en la estructura de la tabla.
- Sintaxis:
  - CONVERT (Campo, Nuevo Tipo);
- Ejemplo:
  - CONVERT (Cantidad, INT);
  - CONVERT (Portal, INT);

GROUP BY.

- Permite agrupar los datos repetidos mediante una columna determinada.
- GROUP BY se incluye después de WHERE y antes de ORDER BY o HAVING, si se utilizan también estas sentencias.
- Sintaxis:
  - SELECT <Nombre del Campo 1, Nombre del Campo 2, Nombre del Campo N> FROM Nombre de la Tabla1
  - [WHERE Condición;]
    GROUP BY <Campo de Agrupamiento 1, Campo de Agrupamiento 2, …, Campo de Agrupamiento N>;
- Ejemplos:
  - SELECT portal FROM Morosos GROUP BY portal;
  - SELECT Nombre, Concepto FROM Morosos
    WHERE Portal = "10"
    GROUP BY Concepto;

CONSULTAS DE TOTALES.

- Se usan funciones de agregado y la sentencia GROUP BY.
- Estas consultas agrupan campos o columnas y hacer cálculos para dichos grupos.
- Para el cálculo se necesita una función de agregado y una cabecera para la columna calculada.
- Se pueden incluir varias funciones en una consulta.
- Sintaxis:
  - SELECT <Campo a Mostrar>, <Función de Agregación> (<Campo de Cálculo>) AS “Nombre Columna Calculada” FROM <Nombre de la Tabla>
    GROUP BY < Campo de Agrupamiento>

  - SELECT <Campo a Mostrar>, <Función de Agregación> (<Campo de Cálculo>) AS “Nombre Columna Calculada” FROM <Nombre de la Tabla>
    [WHERE <Condición>]
    GROUP BY < Campo de Agrupamiento>

- Ejemplos:
    - SELECT Portal, SUM (Cantidad) AS “Total Debido” FROM Morosos
      GROUP BY Portal; // Total debido agrupado por portales.
    - SELECT concepto, AVG (Cantidad) AS 'Media Cantidades” FROM Morosos
      WHERE Portal = “15”
      GROUP BY Concepto; // Media por cada concepto debido en el portal 15.

HAVING.

- Permite especificar una condición para una función de agregado, es decir, permite mostrar los datos agrupados que cumplan una condición.
- Es independiente de WHERE y se pone a continuación de GROUP BY y antes de ORDER BY, si este último se incluye.
- Sintaxis:
    - SELECT <Campo a Mostrar>, <Función de Agregación> (<Campo de Cálculo>) AS “Nombre Columna Calculada” FROM <Nombre de la Tabla>
      GROUP BY < Campo de Agrupamiento>
      HAVING Condición; // En la condición se incluye la función de agregado o un alias suyo.
    - SELECT <Campo a Mostrar>, <Función de Agregación> (<Campo de Cálculo>) AS “Nombre Columna Calculada” FROM <Nombre de la Tabla>
      [WHERE <Condición>]
      GROUP BY < Campo de Agrupamiento>
      HAVING Condición; // En la condición se incluye la función de agregado o un alias suyo.
- Ejemplo:
    - SELECT concepto, SUM (CANTIDAD) AS 'TOTAL DEBIDO'
      FROM MOROSOS
      GROUP BY Concepto
      HAVING SUM (cantidad) > 500.00; // Se suman las cantidades, pero sólo se muestran aquellas en las que el total este por encima de 500.00

OTRAS CONSULTAS CON CALCULOS.

- Se pueden incluir cálculos simples utilizando expresiones aritméticas con operadores matemáticos en una consulta.
- Se incluyen en la zona de campos de la consulta, creándose una nueva columna a la que se puede aplicar un nombre.
- No se agrupan campos, los cálculos son para cada registro.
- Sintaxis:
    - SELECT <Campo, Campos o Todos los campos>, <Campo con Expresión de cálculo> AS “Nombre Columna Calculada” FROM <Nombre de la Tabla>;
      [WHERE <Condición>]
- Ejemplo:
    - SELECT *, (CANTIDAD * 0.90) AS “Cantidad reducida un 10%” FROM MOROSOS;

CONSULTAS EN TABLAS COMBINADAS.
INNER JOIN.

- Permite consultar 2 o más tablas a través de campos comunes, que suelen ser las claves (PK, FK) de las tablas.
- Conecta dos tablas o más tablas a partir de uno o varios campos comunes.
- El resultado producido es la unión interna o intersección de 2 o más tablas.
- Sintaxis:

  - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla 1> INNER JOIN <Tabla 2>
    ON <Tabla 1.Campo (Con la PK)> = <Tabla 2.Campo (Con la FK)>;
    [WHERE Condición;]

  - **Opción Equivalente sin INNER JOIN. (Producto cartesiano con condición).**
    - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla 1, Tabla 2>
      WHERE <Tabla 1.Campo (Con la PK)> = <Tabla 2.Campo (Con la FK)>;
      [AND Condiciones;]
- Ejemplos:
  - SELECT * FROM Vendedores INNER JOIN Productos
    ON Vendedores.Codven = Productos.Codven;

  - SELECT * FROM Vendedores INNER JOIN Productos
    ON Vendedores.Codven = Productos.Codven
    WHERE Departamento = “Electrodomésticos”;

  - SELECT * FROM Vendedores, Productos
    WHERE Vendedores.Codven = Productos.Codven;

  - SELECT * FROM Vendedores, Productos
    WHERE Vendedores.Codven = Productos.Codven
    AND Departamento = “Electrodomésticos”;

INNER JOIN EN 3 O MÁS TABLAS.

- Sintaxis**:**
  - Opción 1.
    - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla 1> INNER JOIN <Tabla 2> INNER JOIN <Tabla 3>
      ON < Tabla 2.Campo PK> = <Tabla3.Campo FK> (O claves a la inversa).
      ON < Tabla 1.Campo PK > = < Tabla 2.Campo FK >
      [WHERE Condición;]
  - Opción 2.
    - SELECT< Campo | Campo1, Campo2, ...Campo N | * > FROM

(<Tabla 1> INNER JOIN <Tabla 2> ON < Tabla 1.Campo PK > = < Tabla 2.Campo FK >)
INNER JOIN <Tabla 3> ON < Tabla 2.Campo PK> = <Tabla3.Campo FK>;

- Opción 3.
  - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla 1>
    INNER JOIN (<Tabla 2>, <Tabla 3>)
    ON (<Tabla2.CodT1 = Tabla1.CodT1 AND Tabla3.CodT2 = Tabla2.CodT2>);

- Ejemplos:

  - SELECT * FROM Vendedores INNER JOIN Pedidos INNER JOIN Productos
    ON Vendedores.Codven = Pedido.Codven
    ON Productos.CodPro = Pedido.CodPro;

  - SELECT * FROM Vendedores INNER JOIN Pedidos INNER JOIN Productos
    ON Vendedores.Codven = Pedido.Codven
    ON Productos.CodPro = Pedido.CodPro
    WHERE Departamento = “Electrodomésticos”;

  - SELECT * FROM (Vendedores INNER JOIN Pedidos
    ON Vendedores.Codven = Pedido.Codven)
    INNER JOIN Productos ON Productos.CodPro = Pedido.CodPro;

  - SELECT * FROM Vendedores INNER JOIN (Pedidos, Productos)
    ON (Vendedores.Codven = Pedido.Codven AND Productos.CodPro = Pedido.CodPro);

LEFT [OUTER] JOIN.

- Permite consultar 2 o más tablas a través de campos comunes, que suelen ser las claves (PK, FK) de las tablas.
- Conecta dos tablas o más tablas a partir de un campo común.
- El resultado producido es la unión interna o intersección de 2 o más tablas, más el contenido de la tabla que está a la izquierda de la instrucción JOIN pero que no está en la de la derecha (no tiene correspondencia con el contenido de la tabla).
- Sintaxis:

  - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla 1> LEFT [OUTER] JOIN <Tabla 2>
    ON <Tabla 1.Campo (Con la PK)> = <Tabla 2.Campo (Con la FK)>;
    [WHERE Condición;]
- Ejemplos:
  - SELECT * FROM Laboratorios LEFT JOIN Medicamentos
    ON Laboratorios.IdLaboratorios = Medicamentos.IdLaboratorios ;

- SELECT * FROM Laboratorios LEFT JOIN Medicamentos
  ON Laboratorios.IdLaboratorios = Medicamentos.IdLaboratorios ;
  WHERE Tipo = ”Antiácidos””;

RIGHT [OUTER] JOIN.

- Permite consultar 2 o más tablas a través de campos comunes, que suelen ser las claves (PK, FK) de las tablas.
- Conecta dos tablas o más tablas a partir de un campo común.
- El resultado producido es la unión interna o intersección de 2 o más tablas, más el contenido de la tabla que está a la derecha de la instrucción JOIN, pero que no está en la de la izquierda.
- Sintaxis:

  - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla 1> RIGHT [OUTER] JOIN <Tabla 2>
    ON <Tabla 1.Campo (Con la PK)> = <Tabla 2.Campo (Con la FK)>;
    [WHERE Condición;]
- Ejemplos:
  - SELECT * FROM Medicamentos RIGHT JOIN Laboratorios
    ON Medicamentos.IdLaboratorios = Laboratorios.IdLaboratorios;
  - SELECT * FROM Medicamentos RIGHT JOIN Laboratorios
    ON Medicamentos.IdLaboratorios = Laboratorios.IdLaboratorios;
    WHERE Tipo = ”Antiácidos””;

FULL [OUTER] JOIN.

- MySQL no soporta este tipo de combinación, por lo que hay que realizarla combinado LEFT y RIGHT en una unión.
- Sintaxis:

  - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla 1> LEFT [OUTER] JOIN <Tabla 2>
    ON <Tabla 1.Campo (Con la PK)> = <Tabla 2.Campo (Con la FK)>;
    [WHERE Condición;]
    UNION
    SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla 1> RIGHT [OUTER] JOIN <Tabla 2>
    ON <Tabla 1.Campo (Con la PK)> = <Tabla 2.Campo (Con la FK)>;
    [WHERE Condición;]

- Ejemplos:
  - SELECT * FROM Laboratorios LEFT JOIN Medicamentos
    ON Laboratorios.IdLaboratorios = Medicamentos.IdLaboratorios
    UNION
    SELECT * FROM Medicamentos RIGHT JOIN Laboratorios
    ON Medicamentos.IdLaboratorios = Laboratorios.IdLaboratorios;

COMBINACION DE CONJUNTOS.
UNION.

- Permite combinar varios SELECT independientes en un único conjunto de resultados.
- El resultado de la unión serán todos registros de ambas, en el orden de la unión y sin repeticiones.
- Las columnas utilizadas en cada SELECT deben ser del mismo tipo, estar en el mismo orden y ser el mismo número.
- El nombre de las columnas puede ser distinto, utilizándose para el encabezado en la lista de resultados los nombres de la primera tabla usada.
- Sintaxis**:**
    - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla1>
      [WHERE Condición]
      UNION [ALL|DISTINCT]
      SELECT < Campo | Campo1, Campo2, ...Campo N | *> FROM <Tabla2>
      [WHERE Condición]

        - **DISTINCT.**
            - Elimina filas repetidas en la unión.
            - Opción predeterminada.
        - **ALL.**
            - Muestra también las filas repetidas en la unión.
- Ejemplos:

    - SELECT Marca, Modelo FROM Coches
      UNION
      SELECT Marca, Modelo FROM Coches_Seat;

    - SELECT Marca, Modelo FROM Coches
      UNION ALL
      SELECT Marca, Modelo FROM Coches_Seat;

    - SELECT Marca, Modelo FROM Coches
      WHERE marca = "Seat"
      UNION DISTINCT
      SELECT Marca, Modelo FROM Coches_Seat
      WHERE marca = "Seat";

INTERSECT.

- Permite combinar varios SELECT independientes en un único conjunto de resultados.
- El resultado de la intersección serán todos registros iguales en ambas tablas.

- Las columnas utilizadas en cada SELECT deben ser del mismo tipo, estar en el mismo orden y ser el mismo número.
- El nombre de las columnas puede ser distinto, utilizándose para el encabezado en la lista de resultados los nombres de la primera tabla usada.
- Sintaxis**:**
    - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla1>
      [WHERE Condición]
      INTERSECT
      SELECT < Campo | Campo1, Campo2, ...Campo N | *> FROM <Tabla2>
      [WHERE Condición]

- Ejemplos:

    - SELECT Marca, Modelo FROM Coches
      INTERSECT
      SELECT Marca, Modelo FROM Coches_Seat;

EXCEPT / MINUS.

- Permite combinar varios SELECT independientes en un único conjunto de resultados.
- El resultado de la diferencia son los registros que están en una tabla, pero no en la otra.
- No admite la propiedad conmutativa por lo que la diferencia entre 2 tablas mostrará resultados distintos según su posición en las consultas.
- Las columnas utilizadas en cada SELECT deben ser del mismo tipo, estar en el mismo orden y ser el mismo número.
- El nombre de las columnas puede ser distinto, utilizándose para el encabezado en la lista de resultados los nombres de la primera tabla usada.
- Sintaxis**:**
    - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla1>
      [WHERE Condición]
      EXCEPT
      SELECT < Campo | Campo1, Campo2, ...Campo N | *> FROM <Tabla2>
      [WHERE Condición]
    - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla2>
      [WHERE Condición]
      EXCEPT
      SELECT < Campo | Campo1, Campo2, ...Campo N | *> FROM <Tabla1>
      [WHERE Condición]
        - Según posición tablas, resultados diferentes.
- Ejemplos:

    - SELECT Marca, Modelo FROM Coches
      EXCEPT
      SELECT Marca, Modelo FROM Coches_Seat

    - SELECT Marca, Modelo FROM Coches_Seat

EXCEPT
SELECT Marca, Modelo FROM Coches;

SUBCONSULTAS.

- **Concepto**.
  - Consulta anidada a otra consulta, es decir, es un SELECT dentro de otro SELECT.
  - A la consulta anidada se la denomina subconsulta.
  - A la que la incluye, consulta principal.
  - La subconsulta puede incluirse en la lista de selección de campos, tras la cláusula FROM o, tras WHERE o HAVING.
- **Características.**
  - Todas obligatoriamente van encerradas entre paréntesis.
  - No pueden contener cláusulas ORDER BY, ni ser la unión de varios SELECT.
  - Se ejecuta para cada fila de la consulta principal.
  - No devuelven columnas, devuelven datos o valores.

- **Referencia externa.**
  - Es una referencia al valor de una columna de la consulta principal dentro de la subconsulta.
  - Ejemplo:
    - SELECT Codven, NombreVendedor,
      (SELECT MIN (Fecha_Venta) FROM Productos WHERE Productos.Codven = Vendedores.Codven) AS “Fecha de alta”
      FROM Vendedores;
- **INCLUSIÓN DE CONSULTAS.**
  - **En la lista de selección de campos.**
    - La consulta no puede devolver ni varias filas ni varias columnas sino error.
    - Sintaxis:
      - SELECT < Campo | Campo1, Campo2, ...Campo N | * > (SUBCONSULTA)
        FROM <Tabla>;
        [WHERE <Condición>;]
    - Ejemplo:
      - SELECT CodVendedor, Nombre, (SELECT SUM(Cantidad) FROM Productos
        INNER JOIN Vendedores
        ON Productos.CodVendedor = Vendedores.CodVendedor;

  - **Tras la cláusula FROM.**

    - Es la menos usada.
    - Funciona como una consulta aparte.
    - Pueden devolver varias filas o valores que funcionan como una tabla para la consulta principal.
    - No se pueden poner referencias externas.

- Sintaxis:
  - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM (SUBCONSULTA);
    [WHERE <Condición>;]

  - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla Alias>, (SUBCONSULTA) Alias;
    [WHERE <Condición>;]
- Ejemplo:
  - SELECT *
    FROM (SELECT * FROM Libros WHERE Genero = "Ciencia Ficción")
    WHERE Editorial = "Anaya";

- **Tras WHERE o HAVING.**
  - Es la opción de inclusión más utilizada.
  - La subconsulta actúa como dato u operando de una condición.
  - Puede ser necesario usarlas cuando los datos están en una tabla, pero para seleccionar sus filas necesitamos un dato que está en otra tabla.
  - Sintaxis:
    - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla>;
      WHERE <Condición con SUBCONSULTA>;
  - Ejemplo:
    - SELECT * FROM Productos
      WHERE Cantidad < (SELECT AVG (cantidad) FROM Productos WHERE Fecha_Venta > "2020-1-10");

- **TIPOS DE SUBCONSULTAS COMO CONDICIÓN DE UNA CONSULTA PRINCIPAL.**
  - **TEST DE COMPARACIÓN CON SUBCONSULTA.**
    - Son las que devuelven un solo valor.
    - Comparan el valor de la fila examinada con un único valor producido por la subconsulta.
    - Genera una columna con una sola fila.
    - Sintaxis:
      - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla>;
        WHERE Campo Operador Relacional (SUBCONSULTA);
    - Ejemplo:
      - SELECT Tienda, Ciudad FROM Tiendas
        WHERE Objetivo_Ventas > (SELECT SUM (Ventas) FROM Empleados
        WHERE Empleados.Tienda = Tiendas.Tienda); // Muestra las tiendas que han superado los objetivos de ventas.
      - El campo Tiendas.Tienda es una referencia externa.

  - **TEST DE PERTENENCIA A CONJUNTO (IN / NOT IN).**
    - Devuelve varios valores.
    - Examina si el valor de un campo en la consulta principal está incluido en la lista de valores producidos por la subconsulta.

- Genera una columna con varias filas.
- Sintaxis:
  - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla>;
    WHERE Campo IN | NOT IN (SUBCONSULTA);
- Ejemplo:
  - SELECT CodEmpleado, Oficina FROM Empleados
    WHERE Oficina IN (SELECT Oficina FROM Oficinas
    WHERE Región = "Norte");

- **TEST DE COMPARACIÓN CUANTIFICADA (ANY | SOME | ALL).**
  - Devuelve varios valores.
  - Es una mezcla del test de comparación y el test de conjuntos. Se compara el valor de un campo con cada uno de los valores producidos por la consulta.
  - Genera una columna con varias filas.
  - Los tipos de datos a comparar deben ser campos compatibles.
  - Tipos:
    - **Any / Some**.
      - Se compara la condición de la consulta principal con alguno de los valores devueltos por la subconsulta.
    - **All.**
      - Se compara la condición de la consulta principal con todos de los valores devueltos por la subconsulta.
  - Equivalencias:
    - = ANY ⟺ IN.
    - <> ANY ⟺ NOT IN.
    - <>ALL ⟺ NOT IN
  - Sintaxis:
    - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla>;
      WHERE Campo ANY | SOME | ALL (SUBCONSULTA);
  - Ejemplo:
    - SELECT Oficina, Ciudad FROM Oficinas
      WHERE Objetivo > ANY (SELECT SUM (Ventas) FROM Empleados GROUP BY oficina);
    - SELECT Titulo FROM Libros
      WHERE Precio> ALL (SELECT Precio FROM Editoriales WHERE Editorial = "Anaya");

- **TEST DE EXISTENCIA (EXISTS).**
  - Son las que manejan varias columnas y testean la existencia.
  - Examina si la consulta produce alguna fila de resultados.
  - Si contiene fila el resultado verdadero, si no es falso.
  - La subconsulta puede tener varias columnas.
  - En general, necesita de una referencia externa.
  - Sintaxis:
    - SELECT < Campo | Campo1, Campo2, ...Campo N | * > FROM <Tabla>;

WHERE EXISTS (SUBCONSULTA);

- Ejemplo:
  - SELECT CodEmp, Nombre, Oficina FROM Empleados
    WHERE EXISTS (SELECT * FROM Oficinas WHERE Región = "Sur" AND Empleados.Oficina = Oficinas.Oficina);

VISTAS.

- Una vista es una tabla virtual, ya que no guarda ningún dato dentro, solo muestra los contenidos en otras tablas.
- Una vista es una consulta guardada que no contiene datos (o una copia física de ellos), sino sólo las especificaciones de ejecución de la vista.
- Se usan por seguridad, porque así sólo se accede a una parte de las tablas.
- **CREACIÓN.**
  - Sintaxis**:**
    - CREATE VIEW <Nombre de la Vista>
      AS
      SELECT < Campo | Campo1, Campo2, ..., Campo N | * > FROM <Tabla1, Tabla2, …, Tabla N>
      [WHERE Condición];
  - Ejemplos:
    - CREATE VIEW Todos AS SELECT * FROM Morosos;
    - CREATE VIEW Piscina AS SELECT * FROM Morosos
      WHERE Concepto = "Piscina";
    - CREATE VIEW Aprobados AS SELECT * FROM Alumnos INNER JOIN Notas
      ON Alumnos.IdAlumno = Notas.IdAlumno
      AND Nota >= 5
      ORDER BY Nota DESC;

- **MODIFICACIÓN.**
  - Modificar implica crear una nueva consulta.
  - Sintaxis**:**
    - ALTER VIEW <Nombre de la Vista> AS Nueva Consulta;
  - Ejemplos:
    - ALTER VIEW Piscina AS SELECT * FROM Morosos
      WHERE Concepto = "Piscina AND Portal ="15";
  - **Cambiar el nombre a una vista**.
    - MySQL no tiene opción para cambiar el nombre a una vista, de modo que, el cambio de nombre debe hacerse eliminando la vista y después creando la misma vista con otro nombre.
    - Sintaxis**:**
      - DROP VIEW <Nombre de la Vista>;
      - CREATE VIEW <Nuevo Nombre de la Vista> AS Misma Consulta;
    - Ejemplos:

- DROP VIEW Todos;
- CREATE VIEW TodosMorosos AS SELECT * FROM Morosos;

- **ELIMINACIÓN.**
  - Sintaxis**:**
    - DROP VIEW <Nombre de la Vista>;
  - Ejemplos:
    - DROP VIEW Todos;
    - DROP VIEW Aprobados;
- **CONSULTA.**
  - Para mostrar el contenido de la vista, se usa SELECT igual que con cualquier otra consulta.
  - Sintaxis**:**
    - SELECT < Campo | Campo1, Campo2, ..., Campo N | * > FROM <Nombre de la Vista>
    - [WHERE Condición];
  - Ejemplos:
    - SELECT * FROM Todos;
    - SELECT * FROM Aprobados WHERE nota > 7;

  - **Mostrar las vistas que hay en una base de datos**.
    - USE MYSQL;
    - SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE='VIEW' AND TABLE_SCHEMA = "Nombre de la Base de Datos";

  - **Mostrar las tablas que hay en una base de datos**.
    - USE MYSQL;
    - SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE='BASE TABLE' AND TABLE_SCHEMA = "Nombre de la Base de Datos";

  - **Mostrar las tablas y vistas que hay en una base de datos**.
    - USE MYSQL;
    - SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = "Nombre de la Base de Datos";
  - Ejemplos:
    - USE MYSQL;
    - SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE='VIEW' AND TABLE_SCHEMA = "Morosos";
    - SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA = "Bd_Mundo";
      as.Oficina);

TRANSACCIONES.

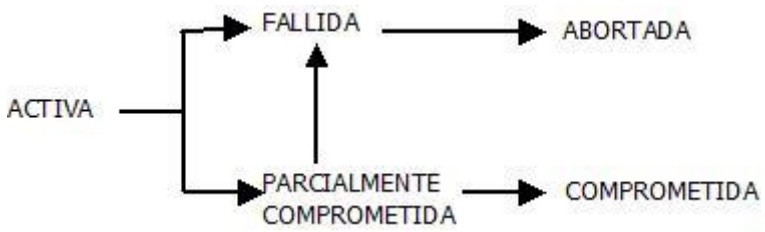
- **Concepto.**

- Conjunto de operaciones que realiza un SGBD para llevar a cabo una unidad lógica de trabajo.
- Es decir, son un conjunto de sentencias SQL que se ejecutan formando una unidad lógica, por lo que se ejecutan, o no, en su totalidad de forma indivisible.

- **Características.**
  - Son indivisibles.
  - Se usan en operaciones de modificación de varias tablas, en el acceso simultáneo por parte de varios usuarios a una misma información.
  - En operaciones de concurrencia garantizan la coherencia e integridad de los datos. Los que se modifican quedan bloqueados para otros usuarios hasta que la transacción no finaliza:
    - Las operaciones INSERT, DELETE o UPDATE generan bloqueos sobre la tabla, por lo que otros clientes no podrán acceder a ella para escribir. Solo podrán hacer lecturas (SELECT), en las que no verán los datos de otro cliente hasta no ser éstos confirmados (COMMIT).
  - Hasta que no se termina, es posible restaurar los datos a su estado inicial. Los cambios de los datos que se están manipulando serán visibles para otros usuarios al terminar la transacción. Existen transacciones que se validan automáticamente.
  - Una transacción termina con un COMMIT para confirma la ejecución de las instrucciones, o en un ROLLBACK para deshacerlas.
- **Inconvenientes.**
  - Pueden aumentar el tiempo de ejecución de las instrucciones o sentencias.
- **Bases de datos que usan transacciones**.
  - En MySQL las tablas que soportan transacciones son las creadas usando el motor de base de datos InnoDB, lo que hace que estas sean más seguras y tolerantes a fallos, ya que, si hay algún problema en el servidor de base de datos, son más fáciles de recuperar.
- **ACID**.
  - Para que un conjunto de instrucciones sea de verdad una transacción debe cumplir las propiedades ACID que tienen las **características** siguientes:
    - **A: ATOMICIDAD:**
      - Cada transacción es una unidad que se da como un todo, si falla parte de ella, todo falla.
    - **C: CONSISTENCIA:**
      - Una transacción va a llevar la base de datos de un estado valido a otro estado valido.
    - **I: AISLAMIENTO:**
      - Una transacción se ejecuta en aislamiento total e independiente de otra transacción.
    - **D: DURABILIDAD:**

- Tras validarse una transacción esta es persistente, ya no se puede volver atrás.

- **Estados de una transacción.**
  - **Transacción activa.**
    - Es el estado de una transacción cuando se inicia o durante su ejecución.
  - **Parcialmente comprometida.**
    - Es el estado alcanzado después de ejecutar la última instrucción.
  - **Fallida.**
    - Estado que asume la transacción cuando no se puede continuar la ejecución normal.
  - **Comprometida.**
    - Estado alcanzado cuando una transacción se completa con éxito.
  - **Abortada.**
    - Estado producido al retroceder en la transacción y restablecer la base de datos a un momento anterior al comienzo de la transacción.



- **Identificador de conexión.**
  - Una sesión de base de datos es una conexión única que empieza al identificarse y termina al salir.
  - Puede haber varias sesiones abiertas por distintos usuarios.
  - Cada usuario de base de datos tiene su propio identificador de conexión asignado por el servidor de base de datos, que se puede conocer mediante la función connection_id().
  - Sintaxis**:**
    - select connection_id();
- **Fallos con las transacciones.**
  - **Error lógico**
    - Es una transacción en el sistema gestor de la Base de Datos SGBD. No se ejecutan normalmente debido a errores internos.
    - Datos que no se encuentran.
    - Entradas incorrectas en formato o valor.
    - Desbordamiento de memoria.
  - **Errores del sistema.**

        - El SGBD se encuentra en un estado que impide que las transacciones puedan ejecutarse normalmente.
    - **Interbloqueo.**
        - Bloqueo permanente de un conjunto de procesos que no pueden avanzar, debido a que dependen de recursos o acciones de otros procesos.
- **Creación de una transacción.**
    - **Iniciar una nueva transacción.**
        - Sintaxis:
            - start / begin transaction;
                - (start en MySQL y MariaDB, begin en SQL Server).
    - **Indicar instrucciones SQL.**
        - Tras comenzar una transacción se especifican todas las sentencias SQL que formarán parte de ella.
        - Para evitar eliminaciones o actualizaciones por error, muchos SGBD tienen por defecto activado el modo seguro (safe update), que sólo permite eliminar o actualizar un registro si se especifica una clave en la cláusula where.
        - Desactivar modo seguro:
            - set sql_safe_updates = 0;
        - Saber si está o no activado el modo seguro:
            - show variables like “sql_safe_updates”; // muestra **OFF** (desactivado) o **ON** (activado).
            - select @@sql_safe_updates; // Muestra **0** (desactivado) o **1** (activado).
        - También, se puede desactivar en los menús de los interfaces visuales de los SGBD, por ejemplo, en Workbench de MYSQL:
            - Edit / Preferences/ SQL Editor/ Desmarcar la casilla Safe Updates.

    - **Validar o confirmar una transacción.**
        - Sintaxis:
            - commit;
    - **Validación automática**.
        - Las transacciones se validan por defecto de forma automática (autocommit).
        - Si se quiere anular autocommit tras iniciar transacción (start transaction):
            - Sintaxis:
                - set autocommit = 0;
    - **Anular o deshacer una transacción**.
        - Sintaxis:
            - rollback;
    - **Punto de parada**.

- Punto, dentro del conjunto de instrucciones SQL, al que se podría volver al anular o deshacer una transacción.
- Sólo se deshacen las acciones o instrucciones posteriores al punto, no las anteriores.
- **Creación de un punto de parada.**
  - Sintaxis:
    - savepoint nombre del punto de parada;
  - Ejemplo:
    - savepoint p1;
- **Vuelta a un punto de parada determinado**.
  - Sintaxis:
    - rollback to punto de parada;
  - Ejemplo:
    - rollback to p1;

USUARIOS.

- Una base de datos puede ser utilizada por distintos usuarios, que deben tener una cuenta de usuario para poder acceder a ellas.
- Es el administrador de base de datos el que se encarga de crearlos y, por cuestiones de seguridad, asignarles los permisos o privilegios que cada uno deba tener.
- Independientemente de que se creen usuarios, posteriormente hay que crear conexiones para que estos usuarios puedan conectarse a los servidores de base de datos. Esto se hace dentro de cada sistema gestor de base de datos de la forma en que éstos lo tengan establecido.
- Componentes de una cuenta de usuario:
  - Nombre del usuario.
  - Nombre del host o servidor.
  - Sintaxis:
    - 'nombre_usuario'@'servidor'
    - Tipos de servidores:
      - **Local**.
        - Localhost
        - 127.0.0.1
        - ::1
      - **Específico**.
        - Dirección IP de dicho servidor.
      - **Cualquier servidor sin restricción**.
        - %
        - Permite acceso al servidor MySQL desde cualquier otro equipo que tenga acceso a él, a través de la red.
- **Usuario predeterminado**.

- MySQL incluye una cuenta de usuario predeterminada:
  - Usuario: root.
  - Cuenta: root@%.
- El usuario predeterminado raíz (root), es el administrador de MySQL y tiene todos los privilegios de la base de datos.
- Es un superusuario, pero inicialmente no tiene contraseña.
- Por seguridad, interesa que *root* tenga una contraseña segura y sólo ser utilizado para operaciones administrativas, porque lo tienen todas las instalaciones de MySQL y cualquier persona con acceso a la instancia de base de datos podría utilizarlo.
- En este sentido, solo dar acceso *root* a los administradores de base de datos que lo necesiten.

- **Usuario anónimo.**
  - Usuario sin nombre, contraseña, ni permisos.
  - En realidad, se le asigna por defecto el permiso *usage.*
  - Posteriormente, se le pueden asignar otros permisos.
  - A este tipo de usuario también es necesario crearle una conexión para acceder al servidor de base de datos. Su conexión no tendrá ni nombre, ni contraseña.
  - Sintaxis:
    - create user “”@”servidor”; // Sin nombre, ni contraseña.
  - Ejemplo:
    - create user “”@”localhost”; // Usuario anónimo creado para el servidor local.
- **Creación**.
  - Sintaxis:
    - create user “nombre _usuario”@”servidor” identified by “contraseña”;
  - Ejemplo:
    - create user “luis”@”localhost” identified by “1234”; // Usuario creado para el servidor local.
    - create user “cristina”@”192.168.200.3” identified by “abcd”; // Usuario creado para un servidor remoto con dirección IP *192.168.200.3*.
- **Asignación de permisos.**
  - Para que un usuario pueda conectarse a un servidor y así usar sus bases de datos, debe tener los permisos o privilegios correspondiente.
  - Se pueden asignar uno, varios o todos los permisos a un mismo usuario.
  - Sintaxis:
    - grant <Permisos> on <Bases de datos> to ‘nombre_usuario’@’servidor’;
    - **Permisos** (algunos, los más utilizados).
      - **super.**

        - Administración y control total sobre el servidor con posibilidad de configurarlo y cambiar su modo de funcionamiento. También permite cerrar la conexión a otros usuarios en otras funciones.
    - **create.**
        - El usuario puede crear bases, tablas, vistas etc.
    - **create user.**
        - Permite crear, modificar, borrar, renombrar y quitar los permisos de los usuarios.
    - **alter**.
        - El usuario puede modificar bases, tablas, vistas, usuarios, etc.
    - **drop**.
        - El usuario puede eliminar bases, tablas, vistas, usuarios, etc.
    - **delete**.
        - El usuario puede borrar registros en una base de datos.
    - **insert**.
        - El usuario puede insertar registros en una base de datos.
    - **select**.
        - El usuario puede realizar consultas en una base de datos.
    - **update**.
        - El usuario puede actualizar registros en una base de datos.
    - **execute**.
        - El usuario puede ejecutar procedimientos almacenados y funciones.
    - **usage**.
        - Solo permite al usuario conectase al servidor de base de datos.
        - No da ningún tipo de privilegio.
    - **file**.
        - Acceso a archivos (lectura y escritura), alojados en el servidor.
    - **grant option**.
        - El usuario puede otorgar permisos y privilegios a otros usuarios.
        - Sintaxis:
            - with grant option.
    - **all | all privileges**.
        - El usuario tiene todos los permisos, excepto grant option.
- **Bases de datos**.
    - **Base de datos.tabla.**

        - Permisos para una tabla determinada de la base de datos.
        - Otorga permisos a nivel de tabla.
    - **Base de datos.vista.**
        - Permisos para una vista determinada de la base de datos.
        - Otorga permisos a nivel de vista.
    - **Base de datos. ***
        - Permisos para todo el contenido de una base de datos determinada.
        - Otorga permisos a nivel de base de datos.
    - ***.***
        - Permisos para todas las bases de datos alojadas en el servidor y su contenido.
        - Otorga permisos a nivel global.
        - Si al usuario se le asignan todos los privilegios, podría con esta opción, crear, modificar o eliminar bases de datos en el servidor.
    - Ejemplos:
        - Ventas.vendedores // Permisos sólo para la tabla *vendedores* de la base *Ventas*.
        - Ventas.* // Permisos para todas las tabla de la base *Ventas*.
        - *.* // Todas las bases - todas las tablas.
- Ejemplos:
    - grant select on Vehiculos.coches to 'luis'@'localhost'; // *Luis* solo puede consultar la tabla *coches* de la base de datos *Vehículos*.
    - grant select, insert, delete, update on Vehiculos.* to 'rosa'@'localhost'; // *Rosa* puede realizar operaciones LMD con todas las tablas de la base de datos *Vehículos*.
    - grant create, drop, select on Vehiculos.* to 'antonio'@'localhost'; // *Antonio* puede crea objetos y eliminarlos dentro de la base *Vehículos*, así como, consultar todas las tablas de dicha base.
    - grant all privileges on Vehiculos.* to 'cristina'@'localhost'; // *Cristina* realizar cualquier operación sobre el contenido de la base de datos *Vehículos*.
    - grant all on *.* to 'eugenia'@'localhost' with grant option; // *Eugenia* tiene asignados todos los privilegios sobre todas las base de datos del servidor, además, puede asignar éstos a otros usuarios.
    - grant usage on *.* to 'pepe'@'localhost'; // *Pepe* sólo puede conectarse al servidor de base de datos, no tiene acceso a sus bases.
    -

- **Revocar o remover Permisos.**

- Se pueden retirar permios a u usuaria previamente otorgados con la sentencia Grant.
- La sintaxis es similar a la de Grant, de modo que:
  - Se puede retirar uno, algunos o todos los permisos.
  - Se puede hacer referencia a una tabla de una base, a todas las tablas de una base, o a todas las bases y sus tablas.
- Sintaxis**:**
  - revoke <Permisos> on <Bases de datos> from 'nombre_usuario'@servidor.
- Ejemplos:
  - revoke delete on Vehiculos.coches from 'rosa'@'localhost'; // Retirada a *Rosa* de la posibilidad de eliminar registros en la tabla *coches* de la base de datos *Vehículos*.
  - revoke create, drop on Vehiculos.* from 'antonio'@'localhost'; // *Antonio* ya no puede crear, ni eliminar objetos, dentro de la base *Vehículos.*
  - revoke all on *.* from 'eugenia'@'localhost'; // A *Eugenia* se le retiran todos sus privilegios sobre todas las base de datos del servidor.

- **Modificar un usuario.**
  - Se pueden modificar las características de un usuario como, por ejemplo, su contraseña.
  - Debe hacerse a través de un usuario que tenga los permisos ALL o CREATE USER.
  - Sintaxis:
    - alter user "nombre _usuario"@"servidor" identified by "nueva contraseña";
  - Ejemplo:
    - create user "luis"@"localhost" identified by "1234"; // Usuario creado para el servidor local con una contraseña determinada.
    - alter user "luis"@"localhost" identified by "4321"; // Mismo usuario con la contraseña cambiada.
- **Borrar un usuario.**
  - Se pueden eliminar usuarios, que ya no podrán volver a conectarse al servidor para acceder a sus bases de datos.
  - Para poder borrar un usuario deben tenerse los permisos drop y create, o también, delete en la base de datos mysql.
  - En las versiones más actuales de MySQL, no es necesario borrar antes los permisos que tenga el usuario a eliminar.
  - Se pueden eliminar varios usuarios a la vez con una única instrucción drop user.
  - Sintaxis:
    - drop user 'nombre_usuario'@'servidor';
  - Ejemplos:
    - drop user 'luis'@'localhost'; // Borrado de un único usuario.

        - drop user ‘javier’@’localhost’, ‘carmen’@’localhost’, ‘lidia’@’localhost’; // Borrado de varios usuarios simultáneamente.
- **Refrescar tabla de privilegios.**
    - Si se hacen cambios o actualizan los permisos de un usuario usando las instrucciones update, delete, etc., sobre la tabla de privilegios (mysql.user), es necesario recargar dicha tabla o volver a conectarse al servidor de base de datos para que los cambios tengan efecto.
    - No es necesario recargar si se usan las instrucciones grant o revoke.
    - Sintaxis:
        - flush privileges;

INFORMACIÓN SOBRE USUARIOS.

- MySQL almacena la información sobre los usuarios y sus privilegios en la base de datos *mysql*.
- Dentro de ella hay varias tablas, entre las que se encuentra *user*, que contine la información de los usuarios y sus privilegios.

- **Usuarios conectados.**
    - Para consultar la información sobre los usuarios actualmente conectados al servidor de base de datos, se usa la función current_user().
    - Sintaxis:
        - select current_user();
- **Usuarios conectados y sus estados.**
    - Si se quiere conocer qué usuarios están conectados y cuál es su estado actual (activo, inactivo, etc.), se consulta la base de datos del sistema *information_schema*.
    - Sintaxis:
        - select user, host, command from information_schema.processlist;
- **Usuarios conectados y sus permisos.**
    - **Usuario actual.**
        - Para mostrar los permisos de usuario actual.
        - Sintaxis:
            - show grants.
        - Son equivalentes:
            - show grants for current_user.
            - show grants for current_user().
    - **Usuario especificado.**
        - Se pueden mostrar los permisos de un usuario concreto.
        - Sintaxis**:**
        - show grants for ‘nombre_usuario’@’servidor’.
            - Muestra los permisos o privilegios del usuario especificado.

        - Ejemplo**:**
            - show grants for “eugenia”@”localhost”;
- **Consultas sobre la tabla de privilegios.**
    - **Ver usuarios.**
        - Para mostrar todos los usuarios creados en el servidor de base de datos se usan las siguientes consultas:
            - select user from user; // Necesario previamente poner en uso la base de datos *mysql* (use mysql).
            - select user from mysql.user; // Sin poner en uso la base de datos *mysql*.
    - **Ver usuarios y servidor.**
        - Mostrar todos los usuarios y a qué servidor o dirección IP tienen permiso de acceso:
            - select user, host FROM user; // Poner en uso antes la base de datos *mysql*.
            - select user, host FROM .mysql.user;
    - **Ver toda la información.**
        - Para consultar toda la información relacionada con los usuarios como su nombre, servidor al que se pueden conectar, privilegios otorgados, expiración de contraseñas, etc:
            - select * from user; // Necesario poner en uso antes la base de datos *mysql*.
            - select * from mysql.user;
        - También se puede acceder a esta información por separado indicando, en la lista de campos de la consulta, sólo aquellos campos que se quieran consultar.

PROGRAMACION.