
# Coding Bootcamp

## Sprint 2

# Temario

# Temario

- Base de datos.
- Claves primarias.
- Tipos de datos.
- Instalación de MySQL.
- Acceso a MySQL a través de CLI y GUI.
- Lenguaje SQL.

# Base de Datos

# Base de Datos – Definición

Una base de datos es una **colección de datos que está organizada** de tal manera que pueda ser accedida, gestionada y actualizada con facilidad.

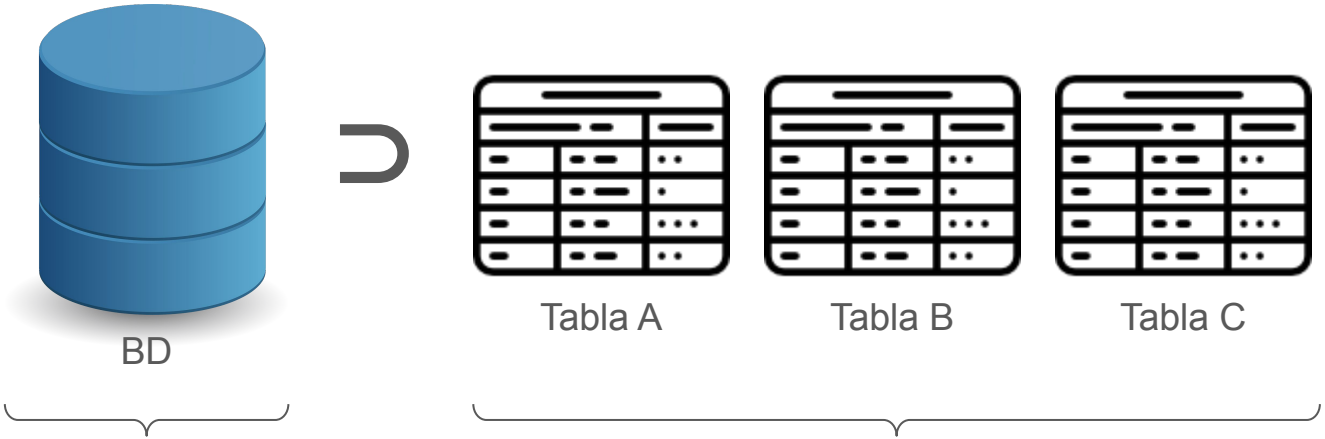
Abreviación: BD o DB.

Notar que la definición no habla de software. De hecho, con esta definición, una guía telefónica o una agenda de contactos también se pueden considerar como bases de datos.

# Base de Datos Relacional (1/2)

Una **base de datos relacional** es una base de datos que organiza la información en una o más **tablas**.

Una **tabla** es una colección de datos organizados en filas y columnas (similar a Excel). A veces, a las tablas se les dice *relaciones*. A las filas también se les llama *registros* o *tuplas*.



# Base de Datos Relacional (2/2)

Ejemplo: **Tabla Usuarios**

# DBMS – Database Management System

Un DBMS es un software que permite crear y manipular bases de datos. Es como una interface entre las BD y las aplicaciones. Algunos gestores populares son:

- MySQL.
- MariaDB.
- SQL Server.
- Microsoft Access.
- Oracle.
- PostgreSQL.
- MongoDB.
- Cassandra.

Relacionales.

No Relacionales.

Los que usaremos en el curso.

¿Relacional vs. No Relacional?
Ver → Link 1. Link 2.

Nota: Al DBMS también se le conoce como *"Motor de BD"*, *"Gestor de BD"* o *"Manejador de BD"*.

# SQL

Es un **lenguaje** para manipular (guardar, obtener, actualizar, borrar) datos de una base de datos relacional.

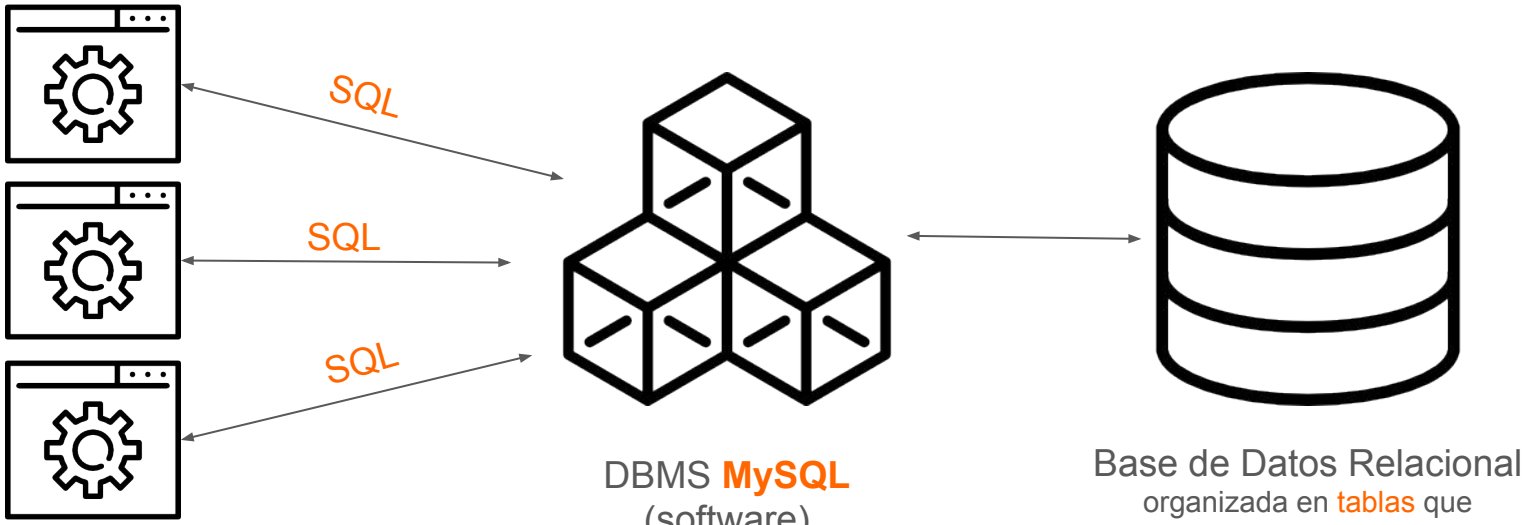
Es el lenguaje utilizado para comunicarse con el gestor de base de datos (DBMS).

SQL = Structured Query Language.

Nota: más adelante en la clase veremos SQL en mayor detalle.

# Node.js + MySQL

En este sprint vamos a crear aplicaciones con **Node.js**, que mediante el lenguaje **SQL** se comunicarán con un gestor de bases de datos llamado **MySQL**, que a su vez será el encargado de manipular la base de datos.



# Clave primaria

# Clave primaria

Es recomendable que en cada tabla exista un campo llamado `id` que permita identificar inequívocamente a cada registro. Se dice que `id` es una clave primaria de la tabla (y sólo puede haber una por tabla).

| id | nombre | apellido | edad | genero |
|---|---|---|---|---|
| 1 | Juan | Pérez | 23 | Hombre |
| 2 | María | Sánchez | NULL | Mujer |
| 3 | Pedro | Álvarez | 28 | Hombre |

Nota: el nombre `id` es simplemente una convención. Podría ser cualquier otro nombre.

Incluso, en el caso de una tabla de usuarios, se podría usar como clave primaria el campo *Cédula de Identidad* ya que el mismo permite identificar inequívocamente a cada persona. Sin embargo, incluso en estos casos es usual tener la columna `id`.

# Tipos de Datos

# Tipos de Datos

Cuando se crea un campo en una base de datos MySQL, se debe especificar el tipo de dato que contendrá dicho campo. Ver documentación.

Según el tipo de dato, también será necesario especificar el largo que tendrá el campo.

Ejemplos:

| | |
|---|---|
| INT (y similares) | Para números enteros. |
| VARCHAR | Para textos cortos. |
| TEXT | Para textos largos. |
| DATETIME | Para fechas/horas. |
| TIMESTAMP | Para fechas/horas. |
| BOOLEAN | Para false/true, se representan con 0 y 1. |

También es posible especificar valores por defecto para cada campo y si el campo admite valores vacíos (NULL).

# Resumen de lo visto hasta ahora

# Resumen de lo visto hasta ahora

- Una base de datos relacional es una base de datos que organiza la información en una o más tablas.
- Una tabla es una colección de datos organizados en filas y columnas.
  En general, las tablas llevan de nombre a sustantivos en plural: `usuarios`, `autos`, `tareas`, etc., aunque existen grandes debates al respecto.
- A las filas se les llama registros o tuplas.
- A las columnas se les llama campos y cada uno tiene su propio tipo.
  Ejemplo: `id`, `nombre`, `apellido`, `cedula`, etc.


- Las tablas suelen tener un campo (o un conjunto de campos), que identifican a cada registro, y se le llama clave primaria. Ejemplos: cédula, nro. de serie, nro. de pasaporte, RUT, ISBN, etc.
- Como clave primaria se suele utilizar un campo llamado `id` que se auto-incrementa cada vez que se crea un nuevo registro.

# Instalar MySQL

# Instalar MySQL (1/2)

Si bien MySQL arrancó siendo un software 100% open-source, la empresa que actualmente detrás del mismo (Oracle) comercializa algunas versiones que son pagas. La versión que usaremos nosotros es la MySQL Community Edition (link) y es gratuita. Por más detalles sobre la instalación entrar aquí.

Links de descarga:

- Windows: https://dev.mysql.com/downloads/installer. Más info aquí.
- Mac: https://dev.mysql.com/doc/refman/8.0/en/osx-installation-pkg.html. También pueden usar Homebrew.
- Linux: https://dev.mysql.com/doc/refman/8.0/en/linux-installation.html.

👉 Si ya tienen instalado un programa como MAMP, XAMPP, WAMP o similar, es probable que ya tengan MySQL (o MariaDB) en sus máquinas, y no sería necesario hacer una instalación adicional.

# Instalar MySQL (2/2)

Por defecto, MySQL estará escuchando en el puerto 3306.

Es probable que en algún momento de la instalación se nos solicite un usuario y contraseña para la base de datos. En un ambiente de desarrollo (nuestra PC) es común poner “root” para ambos datos.

Dependiendo de la forma en que hayan instalado MySQL, a veces es necesario correr el comando `mysql_secure_installation` (en la terminal) para hacer un setup inicial de la base de datos.

Una vez que MySQL se encuentre instalado y corriendo, se podrá interactuar con el mismo mediante el cliente de línea de comandos (CLI) o mediante algún software para administrar bases de datos. Para la primera opción, se deberá escribir en la terminal:

```
mysql -u root -p
```

Si se eligió otro nombre de usuario, sustituir “root” por el mismo.

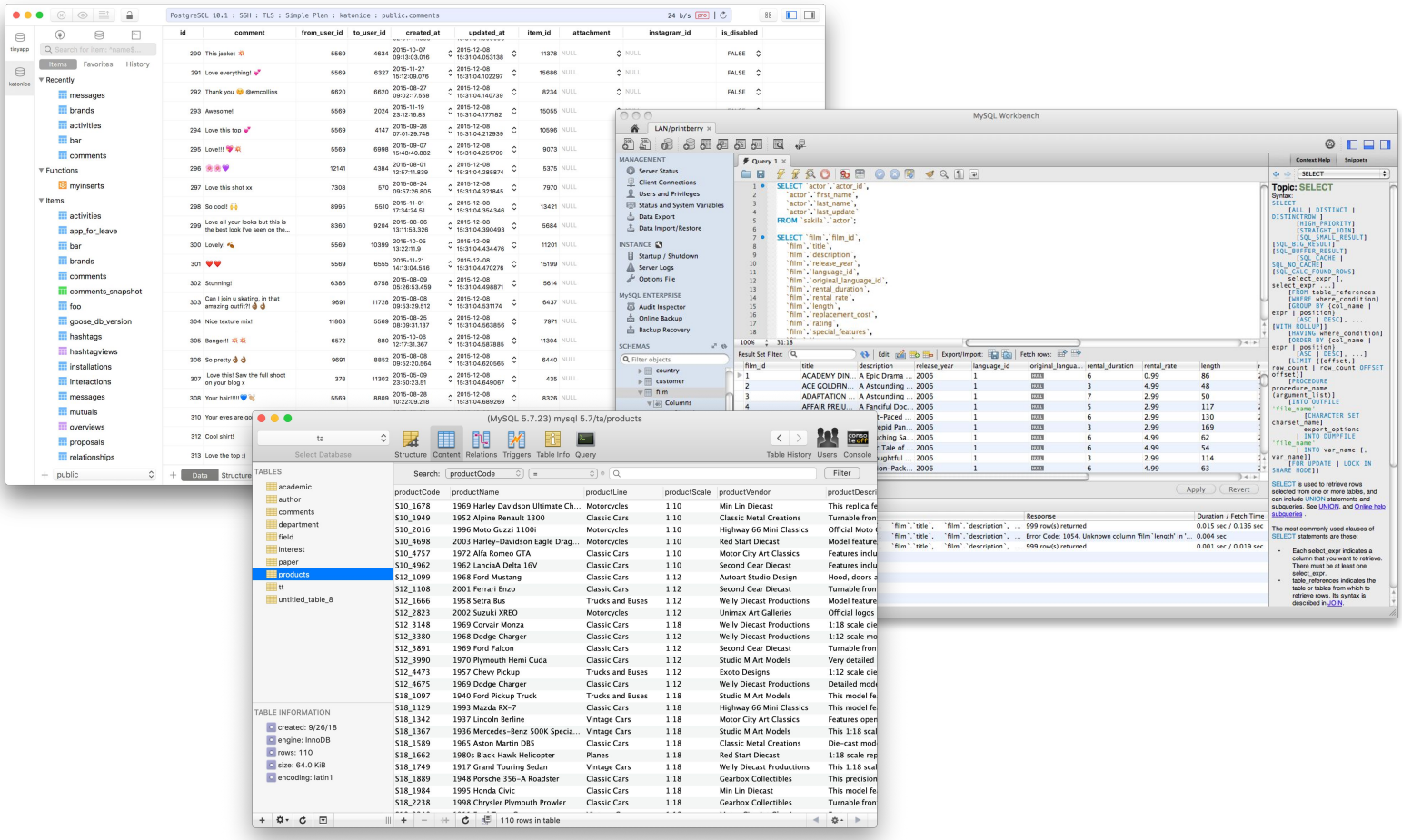
# Aplicaciones para administrar MySQL

# Apps para administrar MySQL

Si bien es posible interactuar con MySQL mediante la CLI, mucha gente prefiere hacerlo a través de una GUI para desktop. Algunas opciones son:

| Nombre | Gratuito | Windows | Mac | Linux |
|---|---|---|---|---|
| TablePlus | Parcialmente | ✔️ | ✔️ | ✔️ |
| MySQL Workbench | Sí | ✔️ | ✔️ | ✔️ |
| DBeaver | Sí | ✔️ | ✔️ | ✔️ |
| Sequel Pro | Sí | ⛔ | ✔️ | ⛔ |
| HeidiSQL | Sí | ✔️ | ⛔ | ⛔ |

También existen GUIs con interfaz web como Node MySQL Admin, phpMyAdmin y Adminer (las dos últimas escritas en PHP), aunque no suelen ser opciones demasiado “robustas”.


PostgreSQL 10.1 : SSH : TLS : Simple Plan : katonice : public.comments
MySQL Workbench
(MySQL 5.7.23) mysql 5.7/ta/products

# SQL

# SQL

Es un lenguaje para manipular (guardar, obtener, actualizar, borrar) datos de una base de datos relacional.

Es el lenguaje utilizado para comunicarse con el gestor de base de datos (DBMS).

SQL = Structured Query Language.

# SQL – Consultas

A cada porción de código SQL que se utiliza para interactuar con la BD se le llama *consulta*.

Ejemplo:

```
SELECT *
FROM usuarios;
```

Esta consulta lo que hace es obtener todos los registros (con todos los campos) de la tabla `usuarios`. El texto en azul son palabras clave de SQL. Lo veremos en detalle a continuación.

Nota: En general hay que tener cuidado al hacer consultas que retornen demasiados registros ya que pueden demorar (y se puede trancar todo).

# SQL – `SELECT` (1/2)

Sintaxis:

```
SELECT columna_1, columna_2, ...,columna_n
FROM nombre_tabla;
```

Toda consulta de tipo `SELECT` retorna como resultado una tabla temporal que contiene los registros resultantes de la consulta.

A esta tabla se le puede llamar "tabla-resultado".

# SQL – SELECT (2/2)

Ejemplo:

```
SELECT id, apellido
FROM usuarios;
```

Retorna:

| id | apellido |
|---|---|
| 1 | Pérez |
| 2 | Sánchez |
| 3 | Álvarez |

Tabla *resultado*; es temporal.

# SQL – Cláusula `WHERE` (1/4)

Sintaxis:

```
SELECT columna_1, columna_2, ...,columna_n
FROM nombre_tabla
WHERE condición;
```

La consulta retorna como resultado una *"tabla-resultado"*, con los registros de `nombre_tabla` que cumplen con la condición.

# SQL – Cláusula `WHERE` (2/4)

Ejemplo:

```sql
SELECT *
FROM usuarios
WHERE apellido = "Pérez";
```

La consulta retorna todos los registros de la tabla `usuarios` que tengan apellido "Pérez".

# SQL – Cláusula `WHERE` (3/4)

Ejemplo:

```sql
SELECT *
FROM usuarios
WHERE telefono = 27065597;
```

La consulta retorna todos los registros de la tabla `usuarios` que tienen asignado el teléfono 27065597.

# SQL – Cláusula `WHERE` (4/4)

Ejemplo:

```sql
SELECT *
FROM usuarios
WHERE birth_date >= "1980-01-01";
```

La consulta retorna todos los registros de la tabla `usuarios` que tienen fecha de nacimiento posterior al 1 de enero de 1980.

# SQL – Operador `AND`

Ejemplo:

```sql
SELECT *
FROM usuarios
WHERE birth_date >= "1980-01-01"
AND birth_date <= "1989-12-31";
```

La consulta retorna los registros de la tabla `usuarios` que nacieron en al década del '80.

# SQL – Operador `OR`

Ejemplo:

```sql
SELECT *
FROM usuarios
WHERE departamento = "Montevideo"
OR departamento = "Canelones";
```

La consulta retorna todos los registros de la tabla `usuarios` que tienen asignado el departamento Montevideo o Canelones.

Es decir, el resultado contiene a todos los usuarios de Montevideo y a los de Canelones.

# SQL – Operador `AND` y `OR`

Ejemplo:

```
SELECT *

FROM usuarios

WHERE birth_date >= "1980-01-01"

AND birth_date <= "1989-31-12"

AND (departamento = "Montevideo" OR departamento = "Canelones" OR
departamento = "San José");
```

La consulta retorna los registros de la tabla `usuarios` que hayan nacido en la década del '80 y que sean de Montevideo, Canelones o San José.

# SQL – Operador `NOT`

Ejemplo:

```sql
SELECT *
FROM usuarios
WHERE NOT departamento = "Montevideo";
```

La consulta retorna los registros de la tabla `usuarios` que no tienen asignado el departamento Montevideo.

Nota: Esta consulta no retorna registros que tengan departamento NULL. Esto es así porque MySQL interpreta el NULL como algo desconocido. Por lo tanto, no se puede decir que un registro con departamento NULL es distinto de "Montevideo".

# SQL – `ORDER BY`

Ejemplo:

```sql
SELECT *
FROM usuarios
ORDER BY birth_date DESC;
```

La cláusula `ORDER BY` permite ordenar los registros de una consulta según determinado campo, de forma ascendente (`ASC`) o descendente (`DESC`), según se indique.

# SQL – `IS NULL`

Ejemplo:

```sql
SELECT *
FROM usuarios
WHERE telefono IS NULL;
```

# SQL – `LIKE`

Ejemplo:

```sql
SELECT *
FROM usuarios
WHERE email LIKE "%@gmail.com";
```

Retorna todos los registros de la tabla `usuarios` cuyo email termine en "@gmail.com".

El símbolo de `%` se usa para indicar que en ese lugar puede ir cualquier conjunto de caracteres.

# SQL – `INSERT INTO` (1/2)

Sintaxis:

```
INSERT INTO nombre_tabla (columna_1, columna_2, ..., columna_n)
VALUES (valor_1, valor_2, ..., valor_n);
```

Esta consulta inserta en la tabla `nombre_tabla` un registro nuevo con los valores indicados.

# SQL – `INSERT INTO` (2/2)

Ejemplo:

```
INSERT INTO usuarios (nombre, apellido, telefono)
VALUES ("María", "González", 24129991);
```

Esta consulta inserta en la tabla `usuarios` un registro nuevo con los valores indicados.