

# Aprendamos MySQL

Introducción a MySQL  
Arquitectura de MySQL  
Instalación  
Primeros pasos

**Roberto Cruzado**

Upgrade-hub  
FullStack Developer  
Bootcamp 10 Weeks

# ¿Qué es MySQL 8?

MySQL es un  
sistema de gestión  
de bases de datos  
relacional.

## SQL (lenguaje de consultas estructuradas).

SQL (Structured Query Language) es un lenguaje de programación estándar e interactivo para la obtención de información desde una base de datos y para actualizarla

Los comandos de SQL se utilizan tanto para consultas interactivas para obtener información de una base de datos relacional y para la recopilación de datos para los informes.

Las consultas toman la forma de un lenguaje de comandos que permite seleccionar, insertar, actualizar, averiguar la ubicación de los datos, y más.

# El gestor de bases de datos más popular

MySQL 8 es la última versión de uno de los sistemas de gestión de bases de datos relacionales más populares del mundo.

Utilizada mayormente por aplicaciones extensamente utilizadas como Wordpress o Joomla

## MySQL

<http://www.mysql.com>

---

## PostgreSQL

Similar a MySQL, pero no tan popular  
<http://www.postgresql.org>

---

## Microsoft SQL Server

Base de datos comercial que requiere  
licencia para su uso

<http://www.microsoft.com/sql-server>

## Oracle Database

También es comercial y requiere licencia  
para su uso

<http://www.oracle.com>

# <http://www.mysql.com>

Punto de partida para la instalación  
Extensiva documentación  
Community Edition es la versión gratuita

# Práctica

Instalación de MySQL en nuestros entornos  
<https://dev.mysql.com/downloads/mysql/>

**Windows**

**MacOS**

**Linux**

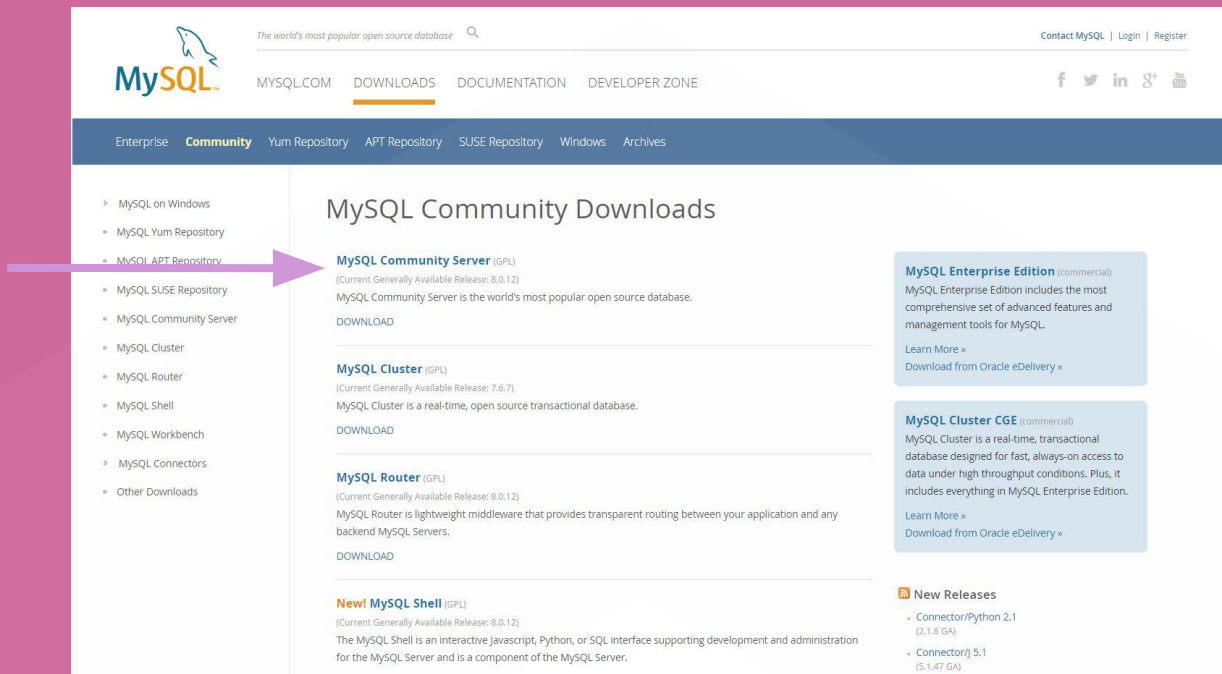
Yum/APT

**Docker**

[https://hub.docker.com/\\_/mysql/](https://hub.docker.com/_/mysql/)

# ¿Qué es MySQL 8?

<https://dev.mysql.com/downloads/>



The screenshot shows the MySQL Community Downloads page. The left sidebar contains a list of download options, with a purple arrow pointing to "MySQL Community Server". The main content area displays details for the MySQL Community Server, including its current release version (8.0.12) and a "DOWNLOAD" link. Other options like MySQL Cluster, MySQL Router, and MySQL Shell are also listed. On the right, there are sections for MySQL Enterprise Edition and MySQL Cluster CGE, each with a "DOWNLOAD" link. At the bottom right, there is a "New Releases" section listing recent updates to the Connector/Python and Connector/J drivers.

**MySQL Community Downloads**

- MySQL on Windows
- MySQL Yum Repository
- MySQL APT Repository
- MySQL SUSE Repository
- MySQL Community Server
- MySQL Cluster
- MySQL Router
- MySQL Shell
- MySQL Workbench
- MySQL Connectors
- Other Downloads

**MySQL Community Server** (GPL)  
(Current Generally Available Release: 8.0.12)  
MySQL Community Server is the world's most popular open source database.  
[DOWNLOAD](#)

**MySQL Cluster** (GPL)  
(Current Generally Available Release: 7.6.7)  
MySQL Cluster is a real-time, open source transactional database.  
[DOWNLOAD](#)

**MySQL Router** (GPL)  
(Current Generally Available Release: 8.0.12)  
MySQL Router is lightweight middleware that provides transparent routing between your application and any backend MySQL Servers.  
[DOWNLOAD](#)

**New! MySQL Shell** (GPL)  
(Current Generally Available Release: 8.0.12)  
The MySQL Shell is an interactive Javascript, Python, or SQL interface supporting development and administration for the MySQL Server and is a component of the MySQL Server.

**MySQL Enterprise Edition** (commercial)  
MySQL Enterprise Edition includes the most comprehensive set of advanced features and management tools for MySQL.  
[Learn More »](#)  
[Download from Oracle eDelivery »](#)

**MySQL Cluster CGE** (commercial)  
MySQL Cluster is a real-time, transactional database designed for fast, always-on access to data under high throughput conditions. Plus, it includes everything in MySQL Enterprise Edition.  
[Learn More »](#)  
[Download from Oracle eDelivery »](#)

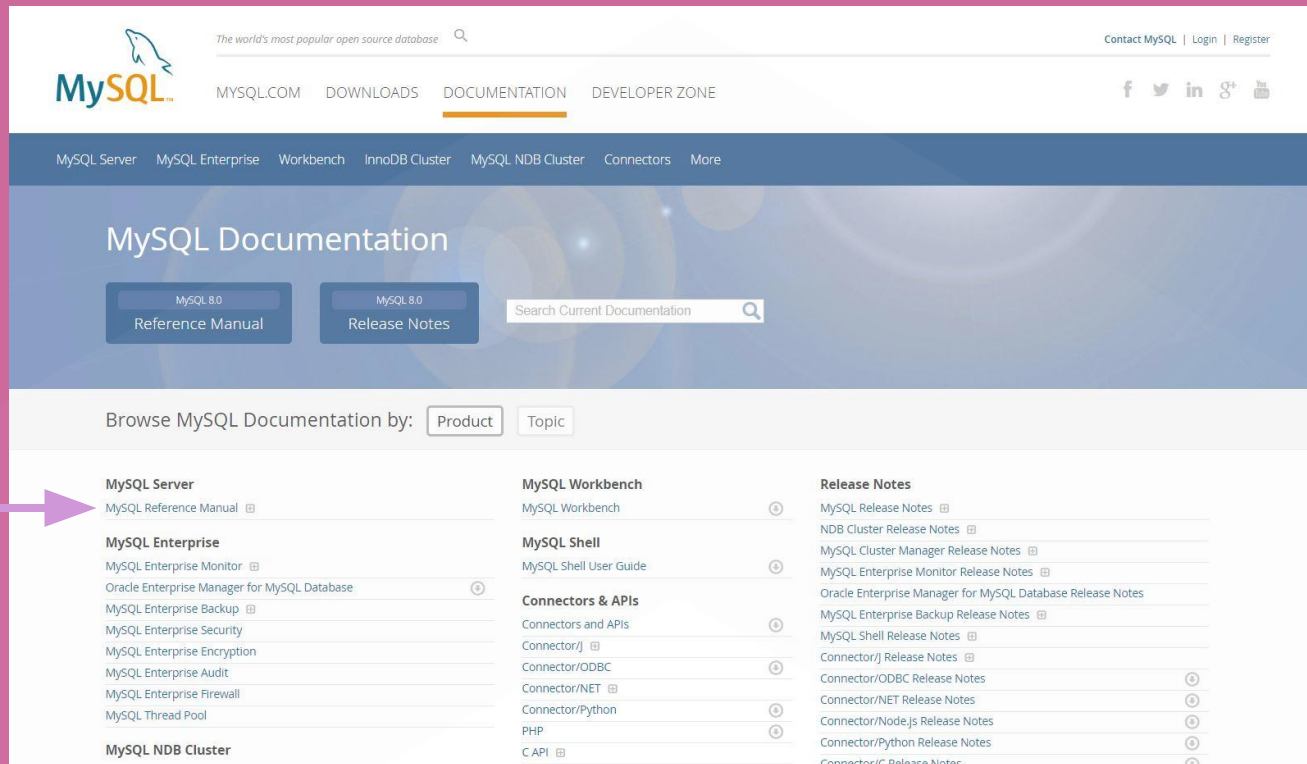
**New Releases**

- Connector/Python 2.1 (2.1.8 GA)
- Connector/J 5.1 (5.1.47 GA)



# ¿Qué es MySQL 8?

<https://dev.mysql.com/doc/>



The screenshot shows the MySQL Documentation website. The header includes the MySQL logo, the tagline "The world's most popular open source database", a search bar, and links for "Contact MySQL", "Login", and "Register". Navigation links for "MYSQL.COM", "DOWNLOADS", "DOCUMENTATION" (highlighted), and "DEVELOPER ZONE" are present. A secondary navigation bar lists "MySQL Server", "MySQL Enterprise", "Workbench", "InnoDB Cluster", "MySQL NDB Cluster", "Connectors", and "More". The main content area is titled "MySQL Documentation" and features buttons for "MySQL 8.0 Reference Manual" and "MySQL 8.0 Release Notes", along with a "Search Current Documentation" bar. Below this, a section titled "Browse MySQL Documentation by:" has tabs for "Product" and "Topic". The "Product" tab is active, showing a list of documentation items categorized by product: MySQL Server, MySQL Enterprise, MySQL NDB Cluster, MySQL Workbench, MySQL Shell, and Connectors & APIs. A purple arrow points to the "MySQL Server" category, which lists the "MySQL Reference Manual". The "MySQL Enterprise" category lists various enterprise-specific documents. The "MySQL NDB Cluster" category lists cluster-related documents. The "MySQL Workbench" category lists the "MySQL Workbench" document. The "MySQL Shell" category lists the "MySQL Shell User Guide". The "Connectors & APIs" category lists various connector and API documents. The "Release Notes" category lists various release notes documents.

# ¿Qué es MySQL 8?

## Generally Available (GA) Releases

## Development Releases

### MySQL Installer 5.7.17

Select Platform:

Microsoft Windows ▼

[Looking for previous GA versions?](#)

Windows (x86, 32-bit), MSI Installer

5.7.17

1.7M

[Download](#)

(mysql-installer-web-community-5.7.17.0.msi)

MD5: df80081cd386da03240c4fb4bae37758 | [Signature](#)

Windows (x86, 32-bit), MSI Installer

5.7.17

386.6M

[Download](#)

(mysql-installer-community-5.7.17.0.msi)

MD5: e03723eb6c6bac271a848bd9031ea859 | [Signature](#)

 We suggest that you use the [MD5 checksums](#) and [GnuPG signatures](#) to verify the integrity of the packages you download.

## Begin Your Download - mysql-installer-community-5.7.17.0.msi

### Login Now or Sign Up for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system
- Comment in the MySQL Documentation

Login »

using my Oracle Web account

Sign Up »

for an Oracle Web account

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

[No thanks, just start my download.](#)

# ¿Qué es MySQL 8?

Seleccionar el tipo de instalación

### Choosing a Setup Type

Please select the Setup Type that suits your use case.

☒ **Developer Default**  
Installs all products needed for MySQL development purposes.

☐ **Server only**  
Installs only the MySQL Server product.

☐ **Client only**  
Installs only the MySQL Client products, without a server.

☐ **Full**  
Installs all included MySQL products and features.

☐ **Custom**  
Manually select the products that should be installed on the system.

Setup Type Description

Installs the MySQL Server and the tools required for MySQL application development. This is useful if you intend to develop applications for an existing server.

This Setup Type includes:

- \* MySQL Server
- \* MySQL Workbench  
The GUI application to develop for and manage the server.
- \* MySQL for Excel  
Excel plug-in to easily access and manipulate MySQL data.
- \* MySQL for Visual Studio  
To work with the MySQL Server from VS.
- \* MySQL Connectors  
Connector/Net, Java, C/C++, ODBC and others.

< Back

Next >

Cancel

# ¿Qué es MySQL 8?

## Definir puerto de comunicación

### Type and Networking

#### Server Configuration Type

Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.

Config Type:

#### Connectivity

Use the following controls to select how you would like to connect to this server.

☒ TCP/IP

Port Number:



☒ Open Firewall port for network access

☐ Named Pipe

Pipe Name:

☐ Shared Memory

Memory Name:

#### Advanced Configuration

Select the checkbox below to get additional configuration page where you can set advanced options for this server instance.

☐ Show Advanced Options

Next >

Cancel

# ¿Qué es MySQL 8?

## Usuarios y Roles

Normalmente para el usuario “root”, la contraseña será “root” o “admin”.

### Accounts and Roles


**Root Account Password**  
Enter the password for the root account. Please remember to store this password in a secure place.

MySQL Root Password:

Repeat Password:

Password Strength: **Weak**

**MySQL User Accounts**  
Create MySQL user accounts for your users and applications. Assign a role to the user that consists of a set of privileges.

MySQL Username	Host	User Role	
 marcosarmiento	%	DB Admin	<div><div>Add User</div><div>Edit User</div><div>Delete</div></div>

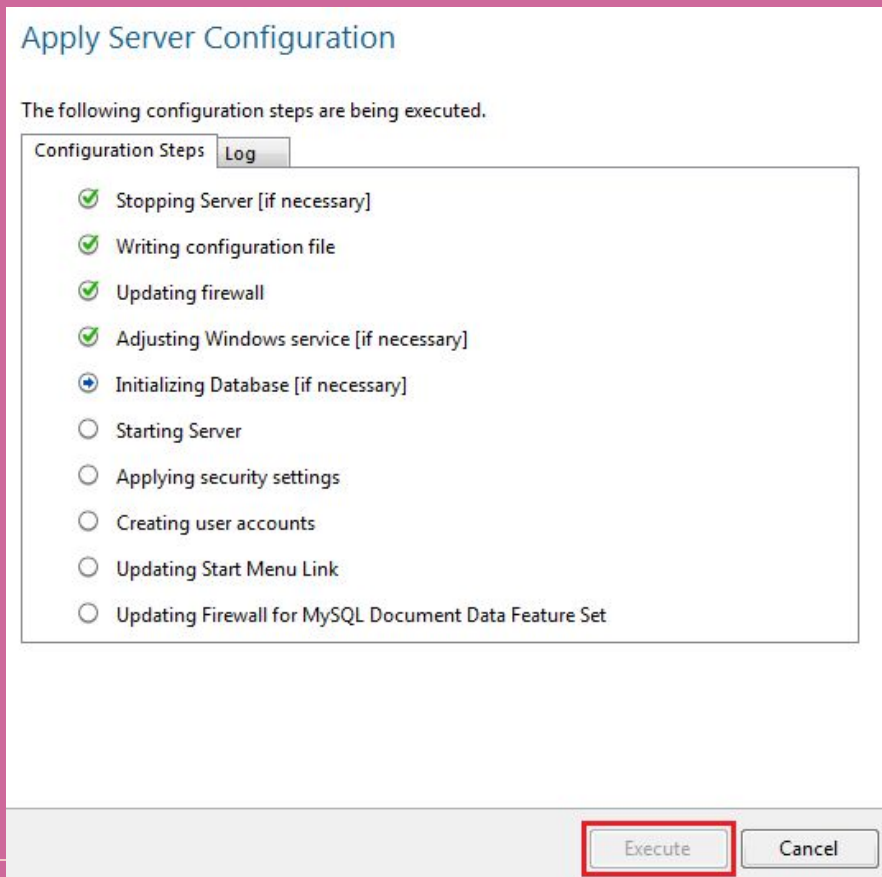
< Back

Next >

Cancel

## Configuración de servicios

Al llegar a esta pantalla debemos dar click en el botón **Execute** para iniciar todos los servicios que hemos creado. Si todo es conforme el sistema podremos proseguir con la instalación.



# ¿Qué es MySQL 8?

Conectando al servidor...

En esta opción nos solicitan que ingresemos el Password que le pusimos al usuarios root. En la parte final del proceso de instalación el sistema verifica que todo los servicios funcionen correctamente y que el usuario root se pueda conectar con el servidor de MySQL.

Connect To Server


Here are the compatible servers installed. If more than one, please select one.

Server	Architecture	Status
MySQL Server 5.7.17	X64	Running

Now give us the credentials we should use (needs to have root privileges). Click check to make sure they work.

User:  Credentials provided in Server configuration

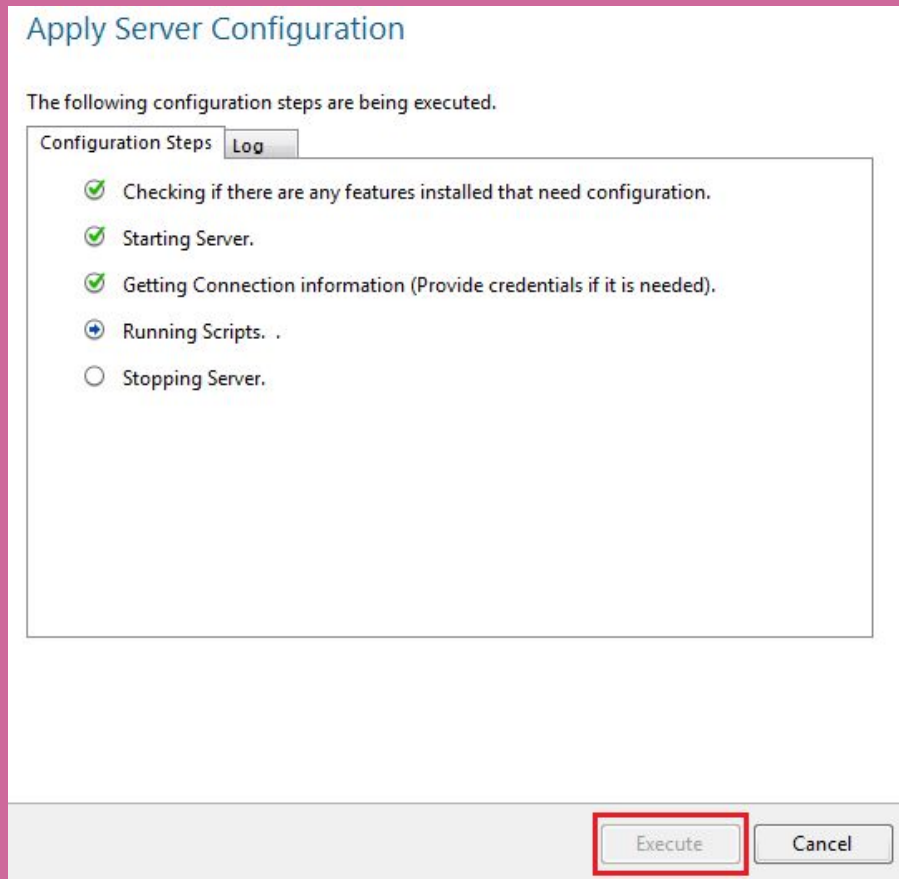
Password:

 Connection successful.



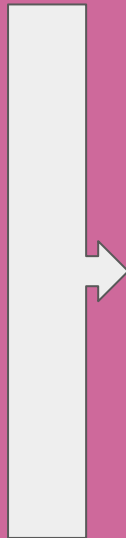
# ¿Qué es MySQL 8?

## Iniciando servicios



# Mostrar Bases de Datos y Tablas existentes

- 1.- Buscar el archivo ejecutable, MySQL 8.0 Command Line Client.
- 2.- Ingresar con el password (normalmente: admin).
- 3.- verificar que el prompt es: Mysql>



```
mysql> show databases;  
  
mysql> use NombreBBDD;  
  
mysql> show tables;  
  
mysql> describe NombreTabla;
```

# Incluir Base de Datos WORLD

(<https://dev.mysql.com/doc/index-other.html>)

1. **Extraer el archivo de instalación a una ubicación temporal como `C:\temp\ o /tmp/`. Desempaquetar el archivo, en un sólo documento que se llame: `world.sql`.**
2. **Situarse en la línea de comandos del cliente MySQL:**

```
shell> mysql -u root -p
```

3. **Ejecutaremos el script `world.sql` para crear la estructura de la base de datos, de la siguiente forma:**

```
mysql> SOURCE C:/temp/world.sql;
```

(Reemplazar `C:/temp/` con el path en donde tengamos descargado el fichero `world.sql` en nuestro sistema)

4. **Confirmar que hemos instalado correctamente esta base de datos de ejemplo con los siguientes comandos...**

# Incluir Base de Datos WORLD (comprobación)

```
mysql> USE world;
```

Database changed

```
mysql> SHOW TABLES;
```

```
+-----+  
| Tables_in_world |  
+-----+  
| city             |  
| country          |  
| countrylanguage |  
+-----+
```

```
mysql> SELECT COUNT(*) FROM city;
```

```
+-----+  
| COUNT(*) |  
+-----+  
| 4079     |  
+-----+
```

```
mysql> SELECT COUNT(*) FROM country;
```

```
+-----+  
| COUNT(*) |  
+-----+  
| 239      |  
+-----+
```

# Descripción de las tablas de la base de datos WORLD

```
| Tables_in_world |
+-----+
| City
| Country
| CountryLanguage
+-----+
3 rows in set (0.00 sec)
```

```
mysql> describe City;
```

```
+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+
| ID         | int(11)   | NO   | PRI | NULL     | auto_increment
| Name       | char(35)  | NO   |     |          |
| CountryCode | char(3)   | NO   | MUL |          |
| District   | char(20)  | NO   |     |          |
| Population | int(11)   | NO   |     | 0        |
+-----+
5 rows in set (0.00 sec)
```

```
mysql> describe Country;
```

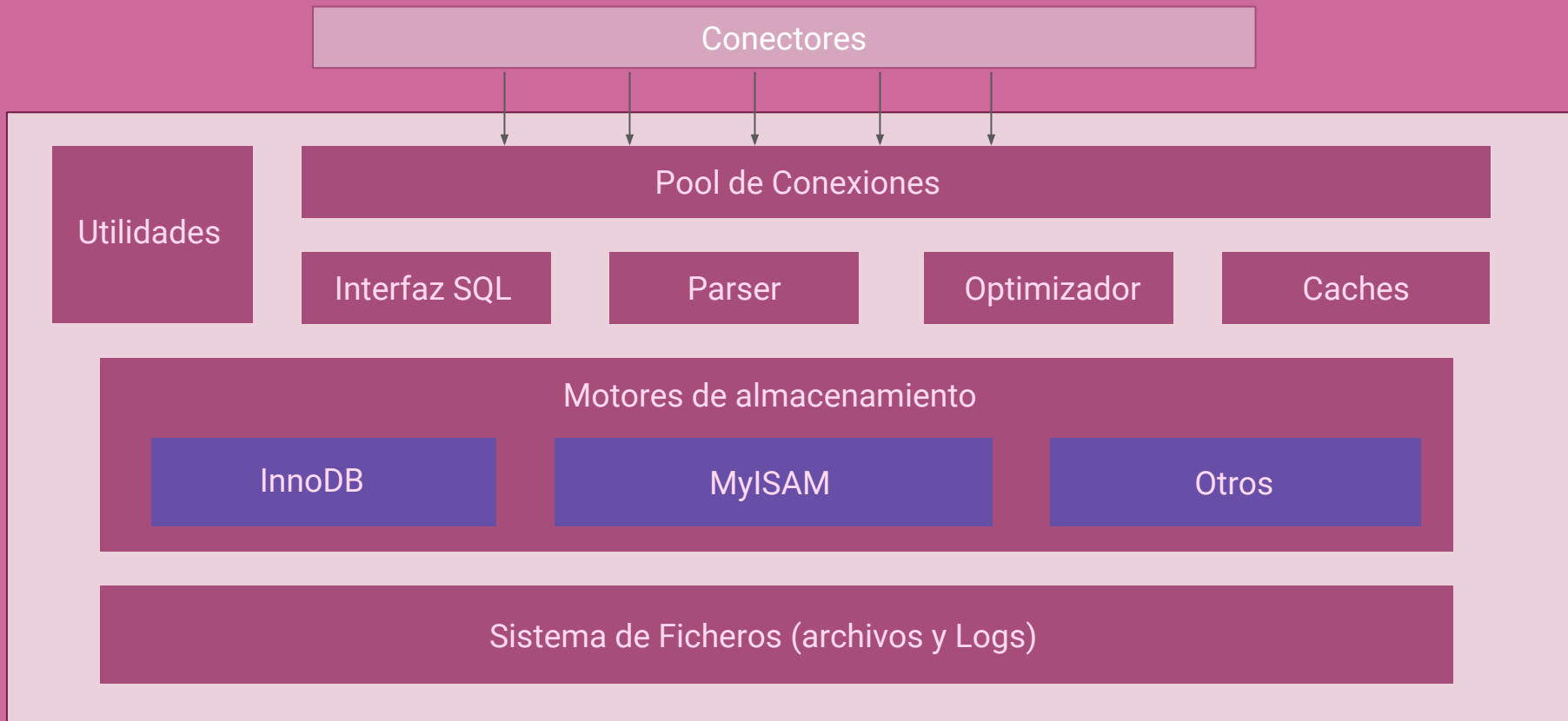
```
+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+
| Code       | char(3)   | NO   | PRI |          |
| Name       | char(52)  | NO   |     |          |
| Continent  | enum('Asia','Europe','North America','Africa','Oceania','Antarctica','South America')
| Region     | char(26)  | NO   |     |          |
| SurfaceArea | float(10,2)
| IndepYear  | smallint(6)
| Population | int(11)   | NO   |     | 0        |
| LifeExpectancy | float(3,1)
| GNP        | float(10,2)
| GNPOld     | float(10,2)
| LocalName  | char(45)  | NO   |     |          |
| GovernmentForm | char(45)
| HeadOfState | char(60)
| Capital    | int(11)   | YES  |     | NULL     |
| Code2      | char(2)   | YES  |     | NULL     |
+-----+
15 rows in set (0.00 sec)
```

```
mysql> describe CountryLanguage;
```

```
+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+
| CountryCode | char(3)   | NO   | PRI |          |
| Language    | char(30)  | NO   | PRI |          |
| IsOfficial  | enum('T','F')
| Percentage  | float(4,1)
+-----+
4 rows in set (0.00 sec)
```

# PRÁCTICA de EJEMPLO sobre la tabla CITY

1. Ver estructura de la tabla
2. Ver todos los registros de la tabla
3. Ver todos los nombres y distritos de las ciudades
4. Ver el nombre de todas las ciudades que tienen el código ESP
5. Sacar la población menor
6. Averigua la suma de todas los habitantes
7. Saca las ciudades del país USA, que su población sea mayor de 10000



**Pool de conexiones:** Los usuarios establecen una conexión a la base de datos, que reside en un hilo de procesamiento, esta conexión ejecuta sentencias en nombre del usuario.

**Parser:** Convierte sentencias SQL en comandos internos

**Optimizador:** Define cómo se ejecuta la consulta y a qué tablas se accede.

**Caches/Buffers:** Mantienen los datos en la memoria



# InnoDB

**Motor de almacenamiento por defecto para MySQL 8 (anteriormente era MyISAM)**

**Capa de caché:** Lugar donde InnoDB almacenará datos en caché para acceder a ellos rápidamente.

**Capa de almacenamiento:** los datos se guardan en archivos de datos

**Capa de transacciones:** Se asegura de que todo se escriba de forma segura en un almacenamiento persistente, incluso cuando el servidor de la base de datos se bloquea.

## MEJORES PRÁCTICAS AL CREAR UNA TABLA

Cada tabla debe tener una clave primaria (PRIMARY\_KEY)

Elige siempre el tipo de dato correcto para cada columna

Elige el tipo de dato más pequeño que se ajuste a tus datos

Crea un índice para cada columna que vaya a ser una condición de un WHERE

# Comandos

## El Lenguaje SQL LDD: Lenguaje de Definición

Sus acciones buscan definir la semántica del esquema relacional: qué relaciones hay, y cómo son, cuáles son sus dominios, cuáles las asociaciones, restricciones, etc.

- Tres acciones básicas: creación, supresión, alteración
- Tres instrucciones básicas: CREATE, DROP, ALTER
- Aplicables a una amplia gama de elementos: tablas, vistas, índices....

## El Lenguaje SQL LMD: Lenguaje de Manipulación

Sus instrucciones constan de: LOCALIZACIÓN + ACCIÓN

Dos tipos de instrucciones: actualización y recuperación:

- Tres acciones de actualización: inserción, borrado, modificación
- Tres instrucciones: **INSERT, DELETE, UPDATE**
- Acción de recuperación: selección
- Instrucciones: **SELECT**

# Lenguaje de Definición

- **CREATE**
- **DROP**
- **ALTER**

## CREATE (databases)

- Una base de datos almacena información mediante un conjunto de tablas, cada una de ellas con una estructura, a priori, diferente..

```
mysql> CREATE DATABASE escuela;
```

La llamada a la variable STATUS, nos indicará, entre otras cosas, la base de datos sobre la que estamos trabajando.

```
mysql> status;
```

## CREATE (table)

- Al crear una tabla debemos resolver qué campos (columnas) tendrá y qué tipo de datos almacenarán cada uno de ellos, es decir, su estructura.
- La tabla debe ser definida con un nombre que la identifique y con el cual accederemos a ella.
- Cada campo (columna) debe tener un nombre. El nombre del campo hace referencia a la información que almacenará.
- Cada campo (columna) también debe definir el tipo de dato que almacenará.
- Si intentamos crear una tabla con un nombre ya existente -->ERROR



## CREATE (table)

```
mysql> CREATE TABLE profesores (  
→ CodProfe VARCHAR(10),  
→ nombre VARCHAR (30),  
→ apellidos VARCHAR (50),  
→ FechaEntrada DATE,  
→ departamento VARCHAR (20),  
→ asignatura VARCHAR(20)  
→ );
```

## TIPOS DE DATOS BÁSICOS

Antes de crear una tabla debemos pensar en sus campos y optar por el tipo de dato adecuado para cada uno de ellos.

- **VARCHAR:** se usa para almacenar cadenas de caracteres.  
Se coloca entre comillas (simples): 'Hola'.  
65535 caracteres  
`varchar(30)` → almacenará una cadena de hasta 30 caracteres.
- **INTEGER o INT:** se usa para guardar valores numéricos enteros, de -2000000000 a 2000000000 aprox.
- **FLOAT:** se usa para almacenar valores numéricos decimales.  
Se utiliza como separador el punto (.).

## OTROS TIPOS DE DATOS

- Fechas y Horas: DATE (fecha), DATETIME (fecha y hora), TIME (hora).
- CHAR (x): define una cadena de longitud fija, su rango es de 1 a 255 caracteres.
- TEXT: bloques de caracteres de mucha longitud para almacenar descripciones, normalmente.
- "NULL". El valor 'null' significa “valor desconocido” o “dato inexistente”. No es lo mismo que 0 o una cadena vacía.
- SET('value1','value2',...). Un campo que puede contener ninguno, uno ó varios valores de una lista. La lista puede tener un máximo de 64 valores

## OTROS TIPOS DE DATOS

- INTEGER tiene subtipos:
  - ❖ MEDIUMINT : va de  $-8000000$  a  $8000000$  aprox. Sin signo va de 0 a  $16000000$  aprox.
  - ❖ SMALLINT : va de  $-30000$  a  $30000$  aprox., sin signo, de 0 a  $60000$  aprox.
  - ❖ TINYINT : define un valor entero pequeño, cuyo rango es de  $-128$  a  $127$ . El tipo sin signo va de 0 a 255.
  - ❖ BOOL o BOOLEAN : sinónimos de `tinyint(1)`. Un valor cero se considera falso, los valores distintos de cero, verdadero.
  - ❖ BIGINT : es un entero largo. Va de  $-9000000000000000000$  a  $9000000000000000000$  aprox. Sin signo es de 0 a  $10000000000000000000$ .

## OTROS TIPOS DE DATOS

- DATE tiene subtipos:
  - ❖ DATE: representa una fecha con formato "YYYY-MM-DD". El rango va de "1000-01-01" a "9999-12-31".
  - ❖ DATETIME: almacena fecha y hora, su formato es "YYYY-MM-DD HH:MM:SS.ssssss". El rango es de "1000-01-01 00:00:00" a "9999-12-31 23:59:59.999999".
  - ❖ TIME: una hora. Su formato es "HH:MM:SS". El rango va de "-838:59:59" a "838:59:59".
  - ❖ TIMESTAMP: Marca temporal en formato 'YYYY-MM-DD HH:MM:SS'.
  - ❖ YEAR(2) y YEAR(4): un año. Su formato es "YYYY" o "YY". Permite valores desde 1901 a 2155 (en formato de 4 dígitos) y desde 1970 a 2069 (en formato de 2 dígitos).

## OTROS TIPOS DE DATOS

- FLOAT (t,d): número de coma flotante. Su rango es de  $-3.4e+38$  a  $-1.1e-38$  (9 cifras).
- DECIMAL o NUMERIC (t,d): el primer argumento indica el total de dígitos y el segundo, la cantidad de decimales.
  - ❖ Si queremos almacenar valores entre 0.00 y 99.99 debemos definir el campo como tipo "decimal (4,2)".
  - ❖ Para los tipos "float" y "decimal" se utiliza el punto como separador de decimales.
- Todos los tipos enteros pueden tener el **atributo "unsigned"**, esto permite sólo valores positivos y duplica el rango ( de 0 a 40000000000 ).
- Los tipos de coma flotante también aceptan el atributo "unsigned", pero el valor del límite superior del rango no se modifica.

## Ejemplos de distintos datos numéricos

- Facturación de una PYME.
- Edad de unos participantes.
- Kilómetros de distancia entre planetas.
- Cantidades de un producto.
- Cuenta total de una compra.
- Usuarios de media en un acceso a evento.
- ...

## Ejemplo a ver:

```
create table visitante(  
  codvisit varchar(10),  
  codTienda varchar(12),  
  nombre varchar(30),  
  edad tinyint unsigned,  
  sexo char(1),  
  domicilio varchar(30),  
  ciudad varchar(20),  
  telefono varchar(11),  
  montocompra float unsigned,  
  primary key (codvisit),  
  foreign key (codTienda) references tienda (codigo)  
);
```



## DROP

- Comando utilizado para eliminar una tabla.

```
mysql> DROP TABLE profesor;
```

- Si quisiéramos borrar una tabla que ya estuviera borrada o directamente una que no existiera en nuestra bbdd → ERROR.

```
mysql> DROP TABLE IF EXISTS profesor;
```

## DROP vs TRUNCATE

- Comando utilizado para eliminar una tabla PRESERVANDO su estructura.

```
mysql> TRUNCATE TABLE profesor;
```

- La diferencia con "drop table" es que esta sentencia borra la tabla, "truncate table" simplemente la vacía.
- La diferencia con "delete" es la velocidad, es más rápido "truncate table" que "delete" (se nota cuando la cantidad de registros es muy grande) ya que éste borra los registros uno a uno.

# PRÁCTICA GUIADA

- Crear una base de datos librería
- Crear tablas para libros, compradores, editoriales (proveedores), pedidos, facturas.
- Elegir los tipos de datos adecuados a cada campo.
- Identificar las “posibles” relaciones entre tablas mediante un DER.
- Visualice las tablas creadas.
- Visualice la estructura de cada una de las tablas y verifique que de verdad era lo que quería indicar, en caso contrario, borre la tabla, y vuelva a crearla.

# PRÁCTICA INDIVIDUAL III (Sol Parcial)

```
drop table if exists libro;  
  
create table libro(  
    cod_libro integer primary key,  
    titulo varchar(40),  
    autor varchar(20),  
    editorial varchar(15),  
    precio float,  
    cantidad integer  
);
```

# ALTER

- Modificar la estructura de una tabla existente:
  - agregar nuevos campos,
  - eliminar campos existentes,
  - modificar el tipo de dato de un campo,
  - (agregar o quitar modificadores como "null", "unsigned", "auto\_increment"),
  - cambiar el nombre de un campo,
  - agregar o eliminar la clave primaria,
  - agregar y eliminar índices,
  - renombrar una tabla.

# ALTER

- Modificar la estructura de una tabla existente:

```
ALTER TABLE [table] ([actions]);
```

actions:

```
ADD COLUMN [columndef] (AFTER) [othercolumn] (FIRST)
```

```
DROP COLUMN [column]
```

```
MODIFY COLUMN [column] [columnnewdef]
```

```
CHANGE COLUMN [column] [columnnewdef]
```

```
RENAME TO [newtablename]
```

# ALTER

```
ALTER TABLE libro ADD cantidad smallint unsigned not null;
```

```
ALTER TABLE libro ADD cantidad2 tinyint unsigned after autor;
```

```
ALTER TABLE libro DROP editorial;
```

Si una tabla tiene sólo un campo, éste no puede ser borrado.

```
ALTER TABLE libro DROP editorial, drop cantidad;
```

Si eliminamos un campo clave, la clave también se elimina

```
ALTER TABLE libro MODIFY cantidad smallint unsigned;
```

```
ALTER TABLE libro MODIFY titulo varchar(40) not null;
```

# ALTER

```
ALTER TABLE libro CHANGE precio coste decimal (5,2);
```

```
ALTER TABLE libro CHANGE titulo nombre varchar(40) not null;
```

```
ALTER TABLE usuario RENAME contacto;
```

```
RENAME TABLE usuario TO contacto;
```



# ALTER

Eliminación de clave antigua e imposición de nueva.

```
ALTER TABLE libro DROP PRIMARY KEY;
```

```
ALTER TABLE libro MODIFY Codtitulo VARCHAR(9) PRIMARY KEY;
```

O más elegantemente:

```
ALTER TABLE libro ADD PRIMARY KEY (CodTitulo);
```

Reordenación de campos:

```
ALTER TABLE libros MODIFY COLUMN numpags int AFTER editorial
```

# PRÁCTICA EN PAREJAS II

- Crear una base de datos Club\_Deportivo
- Identificar (discutir y justificar), tres tablas principales.
- Definir la estructura de cada una de las tablas.
- Argumentar los tipos de datos a utilizar.
- Identificar las “posibles” relaciones entre tablas mediante un DER.
- Visualice las tablas creadas.
- Visualice la estructura de cada una de las tablas y verifique que de verdad era lo que quería indicar, en caso contrario, borre la tabla, y vuelva a crearla.
- Realice modificaciones, borrados, ... y justifique los cambios

# Lenguaje de Manipulación

- INSERT
- DELETE
- UPDATE
- SELECT

# INSERT

- Al ingresar los datos de cada registro debe tenerse en cuenta la cantidad y el orden de los campos.
- Usamos "insert into". Especificamos los nombres de los campos entre paréntesis y separados por comas y luego los valores para cada campo, también entre paréntesis y separados por comas

```
mysql> INSERT INTO usuarios (nombre, clave) VALUES ('Fdo  
Torres','Elniño');
```

- Se pueden ingresar los valores de los campos en distinto orden del indicado en la creación de la tabla, siempre y cuando se indique ESE orden alternativo en el primer paréntesis.

# INSERT

- Dar de alta de forma múltiple

```
mysql> INSERT INTO jugador (nombre, clave) VALUES  
    → ('Fdo Torres', 'El niño'),  
    → ('Christiano Ronaldo', 'El loco'),  
    → ('Iker Casillas', 'El majo'),  
    → ('Leo Messi', 'El peque')  
    → ;
```

## PRÁCTICA de ejemplo

- Dar de alta varios registros en la tabla libro de la base de datos librería

```
mysql> insert into libro (titulo,autor,editorial,precio,cantidad) values ('El niño llorón','Paco','GP',5.50,100);  
mysql> insert into libro (titulo,autor,editorial,precio,cantidad) values ('General Mola','Capitan','Mili',25.75,50);  
mysql> insert into libro (titulo,autor,editorial,precio,cantidad) values ('Juegos de azar','Birgo','JPG',8.20,15);  
mysql> select * from libro;
```

(Como no se dan valores para todas las columnas, el resto se rellena con NULL)

## PRÁCTICA entre alumnos

- Dar de alta varios registros en alguna de las tablas de la base de datos escuela, y polideportivo

# REPLACE

- Cuando intentamos ingresar con INSERT un registro que repite el valor de un campo clave, aparece un mensaje de error indicando que el valor está duplicado. Si empleamos REPLACE en lugar de INSERT, el registro existente se borra y se ingresa el nuevo, de esta manera no se duplica el valor único.

```
INSERT INTO libro (idLibro, titulo, autor, editorial, precio, cantidad) VALUES (5, 'Fisica a diario', 'Aringa', 'tututt', 15.4, 7);
```

```
INSERT INTO libro (idLibro, titulo, autor, editorial, precio, cantidad) VALUES (5, 'SQL para principiantes', 'Edisson', 'cococo', 25.5, 10);
```

En su lugar:

```
REPLACE INTO libro (idLibro, titulo, autor, editorial, precio, cantidad) VALUES (5, 'SQL para ti', 'PepeBotella', 'dolar', 25.5, 10);
```

# DELETE

- Para eliminar los registros de una tabla.
- Borra TODOS los registros de la tabla.

```
mysql> DELETE FROM clientes;
```

- Si queremos eliminar uno o varios registros debemos indicar cuál o cuáles → WHERE condición.

```
mysql> DELETE FROM libros WHERE autor = 'Paco';
```

- Una vez eliminado un registro “no hay forma” de recuperarlo



# UPDATE

- Modificar uno o varios valores de uno o varios registros.

```
mysql> UPDATE profesor SET IDprof='abcdef'
```

El cambio afectará a todos los registros

- Si queremos modificar un solo registro debemos indicar cuál o cuáles → WHERE condición.

```
mysql> UPDATE profesor SET clave='12345' WHERE nombre='Ricardo Menéndez';
```

- Si no encuentra registros que cumplan con la condición del "where", ningún registro es afectado.

```
mysql> UPDATE profesor SET nombre='Jorge Torralba', clave='12345'  
→ WHERE nombre='Ricardo Menéndez';
```

Modificación de  
varios campos

# AUTO\_INCREMENT

- Un campo de tipo entero puede tener otro atributo extra 'auto\_increment'. Los valores de un campo 'auto\_increment', se inician en 1 y se incrementan en 1 automáticamente.
- Se utiliza generalmente en campos correspondientes a códigos de identificación para generar valores únicos para cada nuevo registro que se inserta.
- Sólo puede haber un campo "auto\_increment" y debe ser clave primaria (o estar indexado)

# AUTO\_INCREMENT

- Para definir un campo autoincrementable colocamos "auto\_increment" luego de la definición del campo al crear la tabla.
- Para establecer que un campo autoincrementa sus valores automáticamente, éste debe ser entero (integer) y debe ser clave primaria:

- ```
create table libros(  
  codigo int auto_increment,  
  titulo varchar(50),  
  autor varchar(50),  
  editorial varchar(25),  
  primary key (codigo)  
);
```

- Cuando un campo tiene el atributo "auto\_increment" no es necesario ingresar valor para él, porque se inserta automáticamente tomando el último valor como referencia, o 1 si es el primero

# AUTO\_INCREMENT

- Un campo "auto\_increment" funciona correctamente sólo cuando contiene únicamente valores positivos.
- Está permitido ingresar el valor correspondiente al campo "auto\_increment", por ejemplo:  

```
insert into libros (codigo,titulo,autor,editorial)  
values(6,'Martin Fierro','Jose Hernandez','Paidos');
```
- Pero debemos tener cuidado con la inserción de un dato en campos "auto\_increment". Debemos tener en cuenta que:
  - si el valor está repetido aparecerá un mensaje de error y el registro no se ingresará.
  - si el valor dado saltea la secuencia, lo toma igualmente y en las siguientes inserciones, continuará la secuencia tomando el valor más alto.
  - si el valor ingresado es 0, no lo toma y guarda el registro continuando la secuencia.
  - si el valor ingresado es negativo (y el campo no está definido para aceptar sólo valores positivos), lo ingresa.--> ERROR, debemos definir : `codigo int unsigned auto_increment`

# PRIMARY KEY

- La clave PRIMARY **KEY**, es una columna o varias columnas, que sirven para señalar cuál es la clave primaria de la tabla actual.
- La columna o columnas señaladas no podrán contener valores nulos o repetidos. Será la forma de identificar unívocamente a un registro en particular de es tabla.

```
CREATE TABLE usuario(  
  user_id INT AUTO_INCREMENT PRIMARY KEY,  
  username VARCHAR(40),  
  password VARCHAR(255),  
  email VARCHAR(255) );
```

```
CREATE TABLE departamentos (  
  dep int NOT NULL,  
  departamento varchar(255),  
  PRIMARY KEY (dep) );
```

- Puede definirse como AUTO\_INCREMENT, y la clave será creada automáticamente cada vez que se ingrese un registro.
- Definida en el propio campo o al final.

# PRIMARY KEY

La clave primaria, **PRIMARY KEY**, identifica de manera única cada fila de una tabla.

La columna definida como clave primaria (**PRIMARY KEY**) debe ser UNIQUE (valor único) y NOT NULL (no puede contener valores nulos).

Cada tabla sólo puede tener una clave primaria (**PRIMARY KEY**).

- Podemos tener varias columnas que forman parte de la clave principal:

```
CREATE TABLE Persona (  
  ID int NOT NULL,  
  Apellido varchar(50) NOT NULL,  
  Nombre varchar(25),  
  Age int,  
  PRIMARY KEY (ID,Apellido)  
);
```

```
CREATE TABLE Persona (  
  ID int NOT NULL,  
  Apellido varchar(50) NOT NULL,  
  Nombre varchar(25),  
  Age int,  
  CONSTRAINT PK_Persona PRIMARY KEY (ID,Apellido)  
);
```

Aquí sólo existe una PK, pero su valor se compone del contenido de dos columnas

# PRIMARY KEY

- Para crear una nueva restricción de clave primaria en una columna (ID), cuando la tabla ya está creada:

```
ALTER TABLE Persona ADD PRIMARY KEY (ID);
```

```
ALTER TABLE Persona ADD CONSTRAINT PK_Persona PRIMARY KEY (ID,Apellido);
```

- Eliminación de clave primaria

```
ALTER TABLE Persona DROP PRIMARY KEY;
```

# PRIMARY KEY y AUTO\_INCREMENT

- Para que un campo agregado como clave primaria sea autoincrementable, es necesario agregarlo como clave y luego redefinirlo con MODIFY como AUTO\_INCREMENT. No se puede agregar una clave y al mismo tiempo definir el campo autoincrementable.
- Tampoco es posible definir un campo como autoincrementable y luego agregarlo como clave porque para definir un campo AUTO\_INCREMENT éste debe ser clave primaria.
- Si queremos eliminar la clave primaria establecida en un campo AUTO\_INCREMENT aparece un mensaje de error y la sentencia no se ejecuta porque si existe un campo con este atributo **DEBE ser clave primaria**. Primero se debe modificar el campo quitándole el atributo "auto\_increment" y luego se podrá eliminar la clave.
- Si intentamos establecer como clave primaria un campo que tiene valores repetidos, aparece un mensaje de error y la operación no se realiza.



# FOREIGN KEY

- La clave externa o **FOREIGN KEY**, es una columna o varias columnas, que sirven para señalar cuál es la clave primaria de otra tabla.
- La columna o columnas señaladas como **FOREIGN KEY**, sólo podrán tener valores que ya existan en la clave primaria PRIMARY KEY de la otra tabla.

```
CREATE TABLE departamentos (  
  dep int NOT NULL,  
  departamento varchar(255),  
  PRIMARY KEY (dep)  
)
```

```
CREATE TABLE empleado(  
  per int NOT NULL,  
  nombre varchar(255),  
  apellido1 varchar(255),  
  depto int NOT NULL,  
  PRIMARY KEY (per),  
  FOREIGN KEY (depto) REFERENCES  
  departamentos(dep)  
)
```

# FOREIGN KEY

- Clave primaria de una tabla que se obtiene de una relación triple (una 1:1 y dos 1:N) en un DER, y está formada por dos claves ajenas:

```
CREATE TABLE user_roles(  
  user_id INT NOT NULL,  
  role_id INT NOT NULL,  
  PRIMARY KEY(user_id,role_id),  
  FOREIGN KEY(user_id) REFERENCES users(user_id),  
  FOREIGN KEY(role_id) REFERENCES roles(role_id)  
);
```

# FOREIGN KEY

- Varias FK en una tabla:
  - FOREIGN KEY (nomCampo2) REFERENCES nomTabla (nomCampo1),  
CONSTRAINT FOREIGN KEY (nomCampo2) REFERENCES nomTabla (nomCampo1)
- Modificación:
  - ALTER TABLE empleado ADD FOREIGN KEY (dep) REFERENCES departamentos(dep);

```
mysql> ALTER TABLE pedido ADD FOREIGN KEY (codCli) REFERENCES cliente(codcliente);
```

- Eliminación:
  - ALTER TABLE personas DROP FOREIGN KEY dep;

## SELECT

- Para extraer información de una/varias tablas(s) de una base de datos.

SELECT [\* | nomCol | nomcol1, ..., nomColN] FROM nomTabla [WHERE condición(es)] [opciones]

- Para presentar los valores de una columna determinada de una tabla, usamos:
  - SELECT Nombre\_de\_columna FROM Nombre\_de\_tabla;

```
mysql> SELECT descripcion FROM sucursales;
```

## SELECT

- Para seleccionar más de una columna:
  - `SELECT Nombre_de_columna_1, ..., Nombre_de_columna_N  
FROM Nombre_de_tabla;`

```
mysql> SELECT nombre, direccion, codpostal FROM clientes;
```

- Para seleccionar todas las columnas de una tabla:
  - `SELECT * FROM Nombre_de_tabla;`

```
mysql> SELECT * FROM recursos;
```

## SELECT

- Para utilizar campos existentes y campos calculados:

```
mysql> SELECT importe, 0.10 *importe FROM ventas;
```

La expresión (0.10\*importe) constituye lo que se denomina un campo calculado que se obtiene a partir de un campo/columna de la tabla.

En general es deseable no tener cabeceras de columna complicados como "0.10\*Importe". Para esos casos se usa la instrucción AS

```
mysql> SELECT importe, 0.10 *importe AS Descuento FROM ventas;
```

## WHERE (perteneciente a SELECT)

- Esta cláusula sirve para seleccionar filas, dentro de las columnas seleccionadas. Se pueden seleccionar filas donde una/varias columna tiene(n) un valor determinado.

```
mysql> SELECT * FROM clientes WHERE id=1;
```

```
mysql> SELECT * FROM ventas WHERE fecha < "2010-2-21";
```

## WHERE (perteneciente a SELECT)

Para determinar si un dato es NULL se usa la condición “IS NULL”, para saber si es no nulo se usa: “IS NOT NULL”

```
mysql> SELECT * FROM ventas WHERE sucursal IS NOT NULL;
```



## WHERE (perteneciente a SELECT)

### Operadores de comparación:

- $a = b \rightarrow$  a es igual a b
- $a \neq b \rightarrow$  a es distinto de b
- $a < b \rightarrow$  a es menor que b
- $a > b \rightarrow$  a es mayor que b
- $a \leq b \rightarrow$  a es menor o igual a b
- $a \geq b \rightarrow$  a es mayor o igual a b
- $a \text{ IS NULL} \rightarrow$  a es NULL
- $a \text{ IS NOT NULL} \rightarrow$  a no es NULL

Las condiciones simples pueden aparecer combinadas por operadores lógicos. Los operadores lógicos son AND, OR (seleccionará los registros que cumplan con la primera condición, con la segunda condición o con ambas condiciones), XOR (cumplen 1 de las condiciones, no ambas) y NOT.

El operador NOT requiere paréntesis. Es decir se debe escribir WHERE NOT (salario > 50)

## WHERE (perteneciente a SELECT)

### BETWEEN

Utilizado en lugar de operadores relacionales:

```
select * from libro where precio>=20 AND precio<=40;  
select * from libro where precio BETWEEN 20 and 40;
```

### IN

```
select * from libro where autor='Paco' or autor='Pepe Botella';  
select * from libro where autor IN ('Paco','Pepe Botella');
```

### NOT IN

```
select * from libro where autor NOT IN ('Paco','Pepe Botella');
```

# SELECT - Patrones

el operador "=" (igual), también el operador "<>" (distinto) comparan cadenas de caracteres completas

Para comparar porciones de cadenas utilizamos los operadores "like" y "not like"

```
SELECT * FROM libros WHERE autor LIKE "%Botella%";
```

El símbolo "%" (porcentaje) reemplaza cualquier cantidad de caracteres (incluyendo ningún carácter). Es un carácter comodín.

```
SELECT * FROM libro WHERE titulo NOT LIKE 'F%';
```

## ¿QUÉ DEVUELVE?

Así como "%" reemplaza cualquier cantidad de caracteres, el guión bajo "\_" reemplaza UN carácter.

# SELECT - ORDER BY

Ordenaremos el resultado de un "select" para que los registros se muestren ordenados alfabéticamente por algún campo.

```
SELECT codigo,titulo,autor,editorial,precio FROM libro ORDER BY titulo;
```

si colocamos el número de orden del campo por el que queremos que se ordene:

```
SELECT codigo,titulo,autor,editorial,precio FROM libro ORDER BY 5;
```

Por defecto, si no aclaramos en la sentencia, los ordena de manera ascendente (de menor a mayor). Podemos ordenarlos de mayor a menor, para ello agregamos la palabra clave **DESC**.

# SELECT - ORDER BY

También podemos ordenar por varios campos, por ejemplo, por "titulo" y "editorial":

```
SELECT codigo,titulo,autor,editorial,precio FROM libros ORDER BY titulo, editorial;
```

Podemos ordenar en distintos sentido:

```
SELECT codigo,titulo,autor,editorial,precio FROM libros ORDER BY titulo asc, editorial desc;
```

Obtener de la BBDD city, todas las provincias de España, y las ciudades de cada provincia

```
SELECT district, name, population FROM city where countrycode='ESP' ORDER BY district;
```

# SELECT - GROUP BY

Agrupar registros para consultas detalladas

```
mysql> SELECT ciudad, count(*) AS cantidad FROM cliente GROUP BY ciudad;
```

**Devolverá un listado de ciudades y el total de los clientes de cada una, agrupados por su campo ciudad**

```
mysql> SELECT ciudad, count(telefono) FROM cliente GROUP BY ciudad;
```

**Devolverá los nombres de las ciudades y el total de los clientes con esa procedencia, CON TELEFONO NO NULO, ordenados por su campo ciudad.**

La función COUNT( ) retorna distintos valores cuando enviamos como argumento un asterisco o el nombre de un campo: en el primer caso cuenta todos los registros incluyendo los que tienen valor nulo, en el segundo, los registros en los cuales el campo especificado es no nulo.

# SELECT - GROUP BY

Podemos agrupar por más de un campo, por ejemplo, vamos a hacerlo por "ciudad" y "sexo":

```
SELECT ciudad, sexo, count(*) FROM cliente GROUP BY ciudad,sexo;
```

También es posible limitar la consulta con "where".

```
SELECT ciudad, count(*) FROM cliente WHERE ciudad<>'Madrid' GROUP BY ciudad;
```

Podemos usar las palabras claves "asc" y "desc" para una salida ordenada:

```
SELECT ciudad, count(*) FROM cliente GROUP BY ciudad DESC;
```

# SELECT - HAVING

Permite seleccionar (o rechazar) un grupo de registros ya calculados.

```
mysql> SELECT ciudad, count(*) AS cantidad FROM cliente GROUP BY ciudad;
```

Devolverá un listado de ciudades y el total de los clientes de cada una, ordenados por su campo ciudad

```
mysql> SELECT ciudad, count(*) FROM cliente GROUP BY ciudad HAVING count(*)>100;
```

Devolverá un listado de ciudades y el total de los clientes de cada una, ordenados por su campo ciudad, pero SÓLO aquel grupo de ciudades que supere los cien clientes.

```
mysql> SELECT editorial, avg(precio) FROM libros GROUP BY editorial HAVING avg(precio)>10;
```

## ¿QUÉ DEVUELVE?



# SELECT - HAVING

No debemos confundir la cláusula "where" con la cláusula "having";

- WHERE establece condiciones para la selección de registros de un "select";
- HAVING establece condiciones para la selección de registros de una salida "group by".
- La cláusula group by se utiliza cuando las funciones de agregación se aplican a un grupo de conjuntos de tuplas, y la cláusula having se utiliza para poner una condición a ESOS grupos.
- Si en una misma consulta aparece where y having, se aplica primero el predicado de where

# SELECT - DISTINCT

Se especifica que los registros con ciertos datos duplicados sean obviados en el resultado.

```
mysql> SELECT DISTINCT familia FROM productos;
```

**Sólo devolverá una fila por cada género existente, en la tabla**

# SELECT - DISTINCT

## EJEMPLOS

Listado de las asignaturas de los cursos sin repetición:

```
mysql> SELECT DISTINCT asignatura FROM cursos;
```

Cursos donde la asignatura incluya "BBDD", sin repetir horario ni asignatura:

```
mysql> SELECT DISTINCT horario, asignatura FROM cursos  
→ WHERE asignatura LIKE '%BBDD%';
```

Cantidad de cursos DISTINTOS agrupados por horario:

```
mysql> SELECT horario, count(DISTINCT asignatura) FROM cursos  
→ GROUP BY horario;
```

# SELECT - DISTINCT

# EJEMPLOS

diferencia entre:

```
select district from city where  
countrycode='ESP' order by district;
```

y

```
select distinct district from city where  
countrycode='ESP' order by district;
```

# FUNCIONES DE AGREGADO

También llamadas funciones de agrupamiento, porque operan con sobre conjuntos de registros, no con datos individuales.

Son funciones que toman una colección de valores como entrada y producen un único valor de salida.

- Funciones de agregado más habituales: AVG(), MAX(), MIN(), COUNT(), SUM()

## COUNT (\*)

La función "count()" cuenta la cantidad de registros de una tabla, incluyendo los que tienen valor nulo, si no se especifica columna alguna como parámetro.

```
SELECT COUNT(*) FROM Cliente;
```

```
SELECT COUNT(*) FROM Cliente WHERE Ciudad_Origen= "Madrid";
```

```
SELECT COUNT(Cantidad) FROM Productos  
WHERE nombre_Producto LIKE '%periferico%';
```

Contará aquellos registros que tengan valor NO NULO en su campo "Cantidad"

# SUM (\*), MIN(\*), MAX(\*), AVG (\*)

```
select sum(cantidad) from productos;
```

retorna la suma de los valores que contiene el campo especificado. SOLO datos Numéricos

```
select min(precio) from productos  
where marca='Lasika';
```

retorna el mínimo valor que contiene el campo especificado y cumple la condición impuesta

```
select max(precio) from productos;
```

retorna el máximo valor que contiene el campo especificado

```
select avg(precio) from productos  
where Ref Like '%345%';
```

retorna el valor promedio de los valores del campo especificado. SOLO datos numéricos

# PRÁCTICA

Ejercicios sobre la base de datos tienda (**Ejercicio Extra 1 - Una tabla -**)

- creación de bbddd
- creación de tablas
- inclusión de registros
- realización de actualizaciones sobre la estructura de la tabla
- respuesta en consola de los comandos de consulta necesarios para satisfacer la demanda de información.



## Valores por Defecto

Un valor por defecto se inserta cuando no está presente al ingresar un registro y en algunos casos en que el dato ingresado es inválido.

Para campos de cualquier tipo no declarados "not null" el valor por defecto es "null"

- Cadenas de caracteres → cadena vacía: " " (sin espacio entre medias).
- Valores numéricos → 0;
- En caso de ser "auto\_increment" → valor mayor existente+1 (comenzando en 1).
- Campos de tipo fecha y hora, → 0 (en un campo "date" es "0000-00-00").

## Valores por Defecto

Podemos establecer valores por defecto para los campos cuando creamos la tabla,  
Utilizamos "DEFAULT" al definir el campo.

```
create table Productos1euro(  
  codigo int unsigned auto_increment,  
  nombreP varchar(40) not null,  
  marca varchar(30) DEFAULT " ",  
  modeloP varchar(30) DEFAULT 'Desconocido',  
  cantidad int unsigned not null,  
  precio decimal(4,2) unsigned DEFAULT 99.99,  
  primary key (codigo)  
);
```

Para todos los tipos, excepto  
"blob", "text" y "auto\_increment"  
se pueden explicitar valores por  
defecto con la cláusula "default"

# Funciones Matemáticas

Los operadores aritméticos son "+", "-", "\*" y "/". Todas las operaciones matemáticas retornan "null" en caso de error.

MySQL tiene algunas funciones para trabajar con números:

- ABS(x): retorna el valor absoluto del argumento "x". Ejemplo: **select abs(-20);** retorna 20.
- CEILING(x): redondea hacia arriba el argumento "x". Ejemplo: **select ceiling(12.34);** retorna 13.
- FLOOR(x): redondea hacia abajo el argumento "x". Ejemplo: **select floor(12.34);** retorna 12.
- GREATEST(x,y,...): retorna el argumento de máximo valor.
- LEAST(x,y,...): con dos o más argumentos, retorna el argumento más pequeño.
- MOD(n,m): significa "módulo aritmético"; retorna el resto de "n" dividido en "m". Ejemplos: **select mod(10,3);** retorna 1 . **select mod(10,2);** retorna 0.
- POWER(x,y): retorna el valor de "x" elevado a la "y" potencia. Ejemplo: **select power(2,3);** retorna 8.

# Funciones Matemáticas

- RAND(): retorna un valor de coma flotante aleatorio dentro del rango 0 a 1.0.
- ROUND(x): retorna el argumento "x" redondeado al entero más cercano. Ejemplos: `select round(12.34);` retorna 12. `select round(12.64);` retorna 13.
- SQRT(): devuelve la raíz cuadrada del valor enviado como argumento.
- TRUNCATE(x,d): retorna el número "x", truncado a "d" decimales. Si "d" es 0, el resultado no tendrá parte fraccionaria. Ejemplos: `select truncate(123.4567,2);` retorna 123.45; `select truncate (123.4567,0);` retorna 123.

Todas retornan null en caso de error.

Referencia completa de funciones:

<https://dev.mysql.com/doc/refman/8.0/en/mathematical-functions.html>

## Funciones Matemáticas - EJEMPLOS

```
select nombreP, ceiling(precio), floor(precio) from Producto;
```

```
select nombreP, round(precio) from Producto;
```

```
select nombreP, truncate(precio,1) from Producto;
```

# ¿Qué devuelven?

# COLUMNAS CALCULADAS

Es posible obtener salidas en las cuales una columna sea el resultado de un cálculo y no un campo de una tabla.

```
SELECT nombreP, precioP, cantidadP, precio*cantidad AS Total_Precio FROM Productos;
```

```
SELECT titulo, precio, precio*0.1, precio-(precio*0.1)  
FROM Prodcutos;
```

¿Qué  
devuelve?

# PRÁCTICA

Ejercicios I y II sobre las bases de datos de ejemplo incluidas en MySQL.  
7 preguntas sobre la base de datos SAKILA (1..5, 11 y 12)

Primeras preguntas sobre Articulos-Fabricantes

# Tipo de dato ENUM

El tipo de dato "enum" representa una enumeración.

Es una cadena cuyo **ÚNICO** valor se elige de una lista enumerada de valores permitidos que se especifica al definir el campo.  
Puede tener un máximo de 65535 valores distintos.

Si un "enum" permite valores nulos, el valor por defecto es "null"; si no permite valores nulos, el valor por defecto es el primer valor de la lista de permitidos.

Los tipos "enum" aceptan cláusula "default".



# Tipo de dato SET

Representa un conjunto de cadenas.

Puede tener 1 ó más valores que se eligen de una lista de valores permitidos especificados al definir el campo y separados con comas. Puede tener un máximo de 64 miembros.

Los tipos "set" admiten cláusula "default".

EJEMPLO: un campo definido como **set ('a', 'b') not null**, permite los valores 'a', 'b' y 'a,b'. Si carga un valor no incluido en el conjunto "set", se ignora y almacena cadena vacía.

Es similar al tipo "enum" excepto que puede almacenar más de un valor en el campo.

# Tipo de dato SET

```
create table trabajador(  
  numero_orden int unsigned auto_increment,  
  documento char(8),  
  nombre varchar(30),  
  idioma set('ingles','italiano','francés'),  
  estudios enum('ninguno','primario','secundario',  
  'terciario','universitario'),  
  primary key(numero_orden)  
);
```

```
insert into trabajador (documento,nombre,estudios)  
values('68926315','Rubén Fermizo',5);
```

```
insert into trabajador (documento,nombre,estudios)  
values('95324812','Oscar Acol','PostGrado');
```

```
insert into trabajador (documento,nombre,estudios)  
values('64198351','Mariano Tario', 'universitario');
```

```
insert into trabajador (documento,nombre,idioma)  
values('96284576','Belén Tilla','ingles');
```

```
insert into trabajador (documento,nombre,idioma)  
values('23555444','Elena Nito','ingles,italiano');
```

```
insert into trabajador (documento,nombre,idioma)  
values('79217145','Lola Mento','italiano,ingles,italiano');
```

```
insert into trabajador (documento,nombre,idioma)  
values('22255265','Ana Tomía','polaco');
```

# Tipo de dato SET

```
select * from trabajador where idioma like '%ingles%';  
  
select * from trabajador where idioma='ingles'
```

## ¿Diferencia?

# Declaración de variables

- Las variables nos permitirán almacenar un valor y usarlo en sentencias posteriores.
- Las variables son específicas de cada conexión → una variable definida por un usuario no puede ser vista ni usada por otros y son eliminadas/borradas al abandonar la conexión

Formato : **@**nomVar:=expresión.

```
select @coste_medio:=avg(precioP);  
  
select * from Producto where precio<@coste_medio;
```

# Manejo de fechas:

<https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html>

- **adddate**(fecha, interval expresion): retorna la fecha agregándole el intervalo especificado.

```
select adddate('2006-10-10',interval 25 day) as nuevo_dia;  
select adddate('2006-10-10',interval 5 month) as devolucion
```

- **adddate**(fecha, dias): retorna la fecha agregándole a fecha "dias".:

```
select adddate('2006-10-10',25), retorna "2006-11-04".
```

- **addtime**(expresion1,expresion2): agrega expresion2 a expresion1 y retorna el resultado.

```
select addtime('12:24:50','01:12:26');
```

- **current\_date**: retorna la fecha de hoy con formato "YYYY-MM-DD". Igual que **now()**

- **current\_time**: retorna la hora actual con formato "HH:MM:SS"

- **date\_add**(fecha,interval expresión tipo) .- select date\_add('2006-08-10', interval 1 day);

- **date\_sub**(fecha,interval expresión tipo)

- **datediff**(fecha1,fecha2): retorna la cantidad de días entre fecha1 y fecha2.

```
select datediff('1990-05-24', now()) as Dias_de_vida;
```

# Manejo de fechas:

- **dayname**(fecha): retorna el nombre del día de la semana de la fecha. Ejemplo:

`select dayname('2006-08-10')` o `select dayname(now());`

- **monthname**(fecha): retorna el nombre del mes de la fecha dada.

`select monthname('2006-08-10');`

- **dayofweek**(fecha): retorna el índice del día de semana para la fecha pasada como argumento. Los valores de los índices son: 1=domingo, 2=lunes,... 7=sábado). Ejemplo: `dayofweek('2006-08-10')` retorna 5, o sea jueves.

- **dayofyear**(fecha): retorna el día del año para la fecha dada, dentro del rango 1 a 366. Ejemplo:

`select dayofyear('2006-08-10')`

- **hour**(hora): retorna la hora para el dato dado, en el rango de 0 a 23.

`select hour('18:25:09');`

- **minute**(hora): retorna los minutos de la hora dada, en el rango de 0 a 59.

- **timediff**(hora1,hora2): retorna la cantidad de horas, minutos y segundos entre hora1 y hora2.

# Manejo de cadenas:

<https://dev.mysql.com/doc/refman/8.0/en/string-functions.html>

-**concat**(cadena1,cadena2,...): devuelve la cadena resultado de concatenar los argumentos.

```
select concat('Hola',' ','¿cómo esta?'); select concat(@hora,':',@minuto);
```

-**length**(cadena): retorna la longitud de la cadena enviada como argumento.

```
select length('Hola');
```

- **position**(subcadena in cadena): . Devuelve "0" si la subcadena no se encuentra en la cadena

```
select position('H' in 'UpgradeHub');
```

- **substring**(cadena,posicion,longitud): retorna una subcadena de tantos caracteres de longitud como especifica el tercer argumento, de la cadena enviada como primer argumento, empezando desde la posición especificada en el segundo argumento.

```
select substring('UpgradeHub',8,3);
```

-**replace**(cadena,cadena\_reemplazo,caden\_a\_reemplazar): retorna la cadena con todas las ocurrencias de la subcadena reemplazo por la subcadena a reemplazar.

```
select replace('xxx.mysql.com','x','w');
```

## Consultas sobre varias tablas

```
SELECT city.name AS ciudad, country.name AS pais FROM city, country  
WHERE city.countrycode=country.code ORDER BY pais;
```

```
SELECT articulo.Nombre, Precio, fabricante.Nombre FROM articulo, fabricante  
WHERE articulo.Fabricante = fabricante.CodFabricante;
```

```
SELECT E.Nombre, Apellidos, D.Nombre, Presupuesto  
FROM Empleados E, Departamento D  
WHERE E.Departamento = D.Codigo;
```

¿Qué  
devuelven?



# Subconsultas anidadas

Una subconsulta es una expresión select-from-where que se anida dentro de otra consulta o sentencia.

Se utilizan habitualmente para comprobaciones sobre pertenencia a conjuntos, comparación de conjuntos y cardinalidad de conjuntos

Las subconsultas se DEBEN incluir entre paréntesis

```
SELECT numemp, nombre,  
       (SELECT MIN(fechapedido)  
        FROM pedidos  
        WHERE rep = numemp)  
FROM empleados;
```

```
SELECT numemp, nombre  
FROM empleados  
WHERE contrato = (SELECT MIN(fechapedido)  
                  FROM pedidos)  
ORDER BY nombre;
```

# Subconsultas anidadas

Las subconsultas se emplean cuando una consulta es muy compleja, y se la divide en varios pasos lógicos y se obtiene el resultado con una única instrucción; y cuando la consulta depende de los resultados de otra consulta.

Puede haber subconsultas dentro de subconsultas.

## Subconsulta como expresión:

```
SELECT nombre, precio, precio-(SELECT max(precio)
                                FROM Producto) as diferencia
FROM Producto
WHERE nombre='Taza Rosa';
```

En el ejemplo anterior se muestra el nombre, el precio de un producto y la diferencia entre el precio del producto y el producto más caro.

# PRÁCTICA

Ejercicio Extra 2- Consultas para dos tablas y Anidamiento de consultas  
(Tablas Artículos y Fabricantes)

## Subconsultas anidadas ANY-SOME

Se puede utilizar el predicado **ANY o SOME**, los cuales son sinónimos, para recuperar registros de la consulta principal, que satisfagan la comparación con cualquier otro registro recuperado en la subconsulta. Compara ESCALARES

```
SELECT *  
FROM Producto  
WHERE PrecioUnidad > ANY (SELECT  
    PrecioUnidad FROM DetallePedido  
    WHERE Descuento = 0.25);
```

devuelve todos los productos cuyo precio unitario es mayor que el de cualquier producto vendido con un descuento igual al 25 por ciento:

# Subconsultas anidadas IN - NOT IN

El resultado de una subconsulta con IN (o NOT IN) es una lista.

Después de que la subconsulta devuelva el resultado, la consulta exterior lo usará.

```
select nombre  
from Proveedores  
where codigo IN  
(select codigoProveedor  
from Productos  
where marca='RB');
```

nombre de los proveedores que han traído productos de una determinada la marca RB.

## Subconsultas anidadas ALL

ALL también compara un valor escalar con una serie de valores. Chequea si TODOS los valores de la lista de la consulta externa se encuentran en la lista de valores devuelta por la consulta interna.

```
select nombre,precio
from Producto
where marca='FP' and
      precio > all (select precio
                    from Producto
                    where marca='Dorian');
```

nombre y precio de los productos donde la marca sea FP y su precio sea mayor que cualquiera de los productos de la marca Dorian

# Subconsultas anidadas: CORRELACIÓN

Consultas correlacionadas son aquellas que se ejecutan una detrás de otra dentro de una consulta general (o externa)

La consulta interna se evalúa tantas veces como registros tiene la consulta externa, se realiza la subconsulta para cada registro de la consulta externa.

```
select f.*, (select count(d.numeroitem)
             from Detalles as d
             where f.numero=d.numerofactura) as
             cantidad,
             (select sum(d.preciounitario*cantidad)
             from Detalles as d
             where f.numero=d.numerofactura) as total
from facturas as f;
```

develve una lista de todas las facturas que incluya todos los campos (el número, la fecha, el cliente,...), y además la cantidad de artículos comprados y el coste total:

# Subconsultas anidadas: EXISTS - NOT EXISTS

Cuando se coloca en una subconsulta el operador **EXISTS**, MySQL analiza si hay (o no hay con **NOT EXISTS**) datos que coinciden con la subconsulta.

No se devuelve ningún registro, es como un test de existencia.

MySQL termina la recuperación de registros cuando por lo menos un registro cumple la condición "WHERE" de la subconsulta.

```
SELECT cliente,numero
FROM facturas AS f
WHERE EXISTS
(SELECT * FROM detalles AS d
 WHERE f.numero=d.numerofactura
 AND d.articulo='reloj');
```

devolverá una lista de clientes que compraron el artículo "reloj":



# Subconsultas anidadas con INSERT.

## INSERT con subconsultas

```
INSERT INTO alumnos_becados (documento,nota)  
SELECT (documento,nota) FROM aspirantes;
```

Ingresamos registros en la tabla "alumnos becados" seleccionando registros de la tabla "aspirantes"

La cantidad de columnas devueltas en la consulta debe ser la misma que la cantidad de campos a cargar en el INSERT.

# Subconsultas anidadas con UPDATE y DELETE

## UPDATE y DELETE con subconsultas

```
UPDATE Producto SET precio=precio+(precio*0.1)
WHERE codigoProducto= (select codigo
                        from Proveedor
                        where marca='CLM')
;
```

Actualizamos el precio de todos los productos de la marca CLM

```
DELETE FROM Producto
WHERE codigoProducto= (SELECT pv.codigo
                        from Proveedor as pv
                        where nombre='Ginger');
```

Eliminamos todos los productos del proveedor Ginger

# Subconsultas anidadas con CREATE

CREATE con subconsultas

```
create table Proveedor  
  select distinct codProv as codigoProveedor,  
    nombProv as NombreProveedor  
  from Producto;  
|
```

crea una tabla proveedor,  
rescatando el campo codProv  
de la tabla de Productos

## Subconsultas anidadas - Buenas prácticas -

Siempre se debe probar las subconsultas antes de incluirlas en una consulta exterior, así puede verificar que retorna lo necesario, porque a veces resulta difícil verlo en consultas anidadas.

# Concatenación de tablas

Para evitar la repetición de datos y ocupar menos espacio, se separa la información en varias tablas. Cada tabla tendrá parte de la información total que queremos registrar.

# Concatenación de tablas: JOIN o INNER JOIN

Cuando obtenemos información de más de una tabla decimos que hacemos un "join" (unión).

- Especificamos la condición para enlazarlas con "on".

```
SELECT * from producto JOIN proveedor  
ON producto.codProv = proveedor.cod;
```

Con esta operación se calcula el producto cruzado de todos los registros; así cada registro en la tabla A es combinado con cada registro de la tabla B; pero sólo permanecen (se muestran) en la tabla combinada aquellos registros que satisfacen las condiciones que se especifiquen.

**Este es el tipo de JOIN considerado como predeterminado.**

# PRÁCTICA

- Un club imparte clases de distintos deportes.
  - En una tabla llamada "socios" guarda los datos de sus socios (documento identificativo, nombre y residencia).
  - En una tabla denominada "inscritos" almacena la información necesaria para las inscripciones de los socios a los distintos deportes (deporte, año, cuotaPagada (valores SI o NO) y el documento del socio)
1. Cree las tablas
  2. Ingrese algunos registros para ambas tablas:
  3. Muestre el nombre del socio y todos los campos de la tabla "inscritos":
  4. Muestre el nombre de los socios y los deportes en los cuales están inscritos en el año 2018.
  5. Muestre el nombre y toda la información de las inscripciones del socio con número de documento='222222'.

# PRÁCTICA SOL

```
create table socios (  
  documento char(8) not null,  
  nombre varchar(30),  
  domicilio varchar(30),  
  primary key(documento)  
);
```

```
create table inscritos (  
  documento char(8) not null,  
  deporte varchar(15) not null,  
  año year,  
  matriculaPag enum ('s', 'n'),  
  primary key(documento,deporte,año) );
```



# PRÁCTICA SOL

```
insert into socios values('111111','Juan Perez','Colon 234');  
insert into socios values('222222','Maria Lopez','Sarmiento 465');  
insert into socios values('333333','Antonio Juarez','Caseros 980');
```

```
insert into inscriptos values ('111111','natacion','2017','s');  
insert into inscriptos values ('111111','natacion','2018','n');  
insert into inscriptos values ('222222','natacion','2017','s');  
insert into inscriptos values ('222222','tenis','2018','s');  
insert into inscriptos values ('222222','natacion','2018','s');  
insert into inscriptos values ('333333','tenis','2018','n');  
insert into inscriptos values ('333333','basquet','2018','n');
```

# PRÁCTICA SOL

3.-

```
select s.nombre,i.*  
from socios as s JOIN inscriptos as i  
ON s.documento=i.documento
```

4.-

```
select s.nombre,i.deporte  
from socios as s JOIN inscriptos as i  
ON s.documento=i.documento  
where año=2018;
```

5-

```
select s.nombre,i.*  
from socios as s JOIN inscriptos as i  
ON s.documento=i.documento  
where s.documento='222222';
```

# Concatenación de tablas: LEFT JOIN

Para averiguar qué registros de una tabla no se encuentran en otra tabla.  
QUÉ valores de la primera tabla (de la izquierda) **NO** están en la segunda (de la derecha).

```
select * from proveedor  
LEFT JOIN producto  
ON proveedor.codigo = producto.codprov;
```

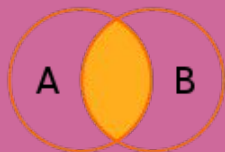
devuelve todos los registros de proveedores, incluso de aquellos de los que no tenemos productos

Un "left join" se usa para hacer coincidir registros en una tabla (izquierda) con otra tabla (derecha); pero, si un valor de la tabla de la izquierda no encuentra coincidencia en la tabla de la derecha, **se genera una fila extra** (una por cada valor no encontrado) con todos los campos puestos a "null".

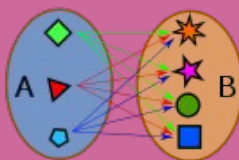
Usando registros de la tabla de la izqda se encuentran registros en la tabla de la dcha.

# Joins del SQL

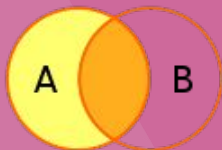
Select (campos)  
From A InnerJoin B  
On A.Clave = B.Clave



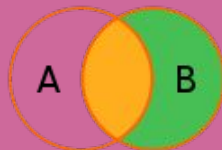
Select (campos)  
From A CrossJoin B



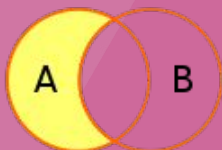
Select (campos)  
From A LeftJoin B  
On A.Clave = B.Clave



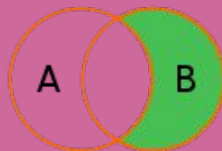
Select (campos)  
From A RightJoin B  
On A.Clave = B.Clave



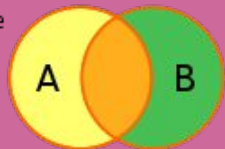
Select (campos)  
From A LeftJoin B  
On A.Clave = B.Clave  
Where B.Clave is Null



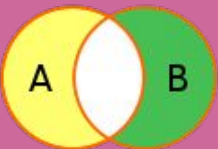
Select (campos)  
From A RightJoin B  
On A.Clave = B.Clave  
Where A.Clave is Null



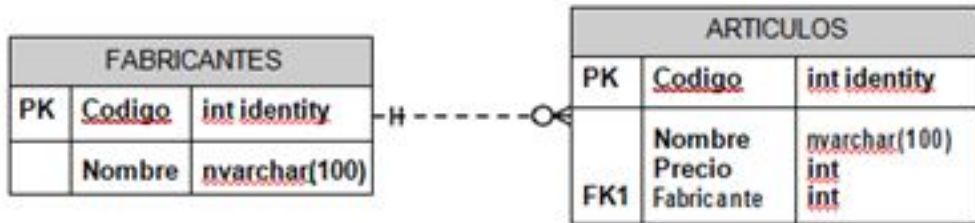
Select (campos)  
From A FullJoin B  
On A.Clave = B.Clave



Select (campos)  
From A RightJoin B  
On A.Clave = B.Clave  
Where (A.Clave is Null) Or (B.Clave is Null)



# Práctica Ejercicios1 con JOINS

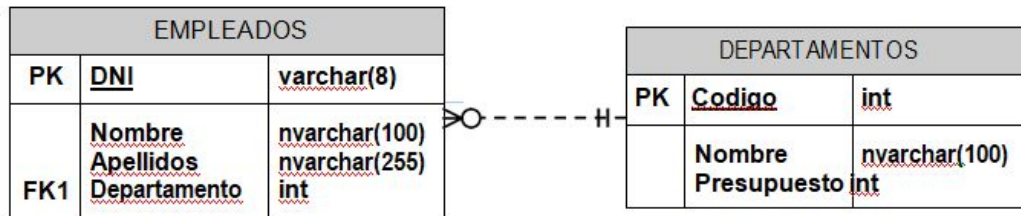


8 preguntas

# Práctica Ejercicios1b con JOINS

## Empleados

|



20 preguntas

# JOIN con más de dos tablas

```
CREATE TABLE peliculas(  
  codigo int unsigned auto_increment,  
  titulo varchar(40) not null,  
  autor varchar(20) default 'Desconocido',  
  PRIMARY KEY (codigo)  
);
```

```
CREATE TABLE abonados(  
  documento char(8) not null,  
  nombre varchar(30),  
  domicilio varchar(30),  
  PRIMARY KEY (documento)  
);
```

```
CREATE TABLE alquileres(  
  documento char(8) not null,  
  codigopelicula int unsigned,  
  fechaprestamo date not null,  
  fechadevolucion date,  
  PRIMARY KEY (codigopelicula, fechaprestamo)  
);
```

# JOIN con más de dos tablas

Al recuperar los datos de los alquileres:

```
select * from alquileres;
```

¿ PROBLEMA ?

Aparecerá el código de la película pero no sabemos el nombre y tampoco el nombre del socio sino su documento. Para obtener los datos completos de cada préstamo, incluyendo esos datos, necesitamos consultar las tres tablas.

```
SELECT nombre,titulo,fechaprestamo  
FROM alquileres as q  
JOIN abonados as a  
ON q.documento=a.documento  
JOIN peliculas as p  
ON q.codigopelicula=p.codigo;
```



# Relaciones de una tabla consigo misma

Una tabla puede relacionarse consigo mismo usando alias para las tablas

| Empleados |         |        |
|-----------|---------|--------|
| Id        | Nombre  | SuJefe |
| 1         | Marcos  | 6      |
| 2         | Lucas   | 1      |
| 3         | Ana     | 2      |
| 4         | Eva     | 1      |
| 5         | Juan    | 6      |
| 6         | Antonio |        |

Queremos obtener un conjunto de resultados con el nombre del empleado y el nombre de su jefe.... ¿CÓMO LO HACEMOS?

```
SELECT Emple.Nombre, Jefes.Nombre  
FROM Empleados Emple, Empleados Jefes  
WHERE  
Emple.SuJefe = Jefes.Id ;
```

# Borrado en cascada con JOIN

Para mantener la integridad referencial en los borrados.

```
DELETE producto, proveedor  
FROM producto pd  
JOIN proveedor pv  
ON pd.codigoproveedor = pv.codigo  
WHERE pv.nombre='CLM';
```

Elimina de la tabla "proveedores" el proveedor CLM y de la tabla "productos" todos los registros con código de proveedor correspondiente a CLM.

Podemos realizar la eliminación de registros de varias tablas (en cascada) empleando DELETE junto al nombre de las tablas de las cuales queremos eliminar registros y después del correspondiente JOIN, colocar la condición WHERE que afecte a los registros a eliminar.

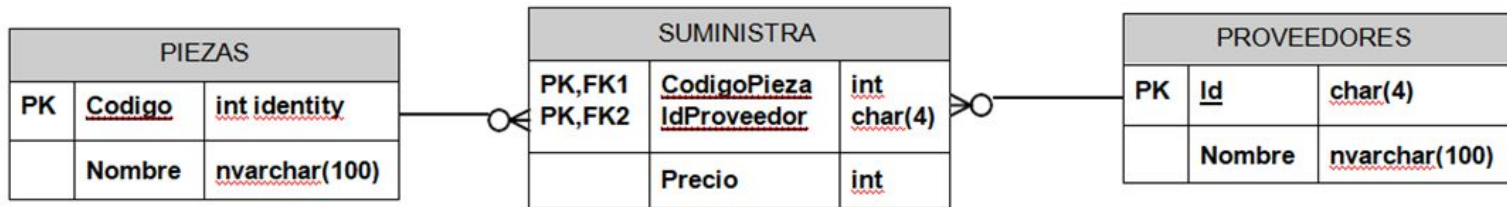
# Práctica Ejercicios2 con JOINS



5 preguntas

# Práctica Ejercicios2b con JOINS

## Almacén



10 preguntas

# ÍNDICES

**USO** → facilitar la obtención de información de una tabla.

- permite encontrar datos rápidamente → localiza registros.
- Una tabla se indexa por un campo (o varios).
- El índice es un tipo de archivo con 2 entradas: un dato (un valor de algún campo de la tabla) y un puntero.
- Posibilitan el acceso directo y rápido haciendo más eficiente las búsquedas. Sin índice, se debe recorrer secuencialmente toda la tabla para encontrar un registro.
- **Útil cuando la tabla contiene miles de registros**
- Es importante identificar el/ los campos por los que sería útil crear un índice, que serán aquellos campos por los cuales se realizan operaciones de búsqueda con frecuencia.

# ÍNDICES

- Una tabla puede tener hasta 64 índices. Los nombres de índices aceptan todos los caracteres y pueden tener una longitud máxima de 64 caracteres.
- Pueden comenzar con un dígito, pero no pueden tener sólo dígitos.
- Una tabla puede ser indexada por campos de tipo numérico o de tipo carácter. También se puede indexar por un campo que contenga valores NULL, excepto los PRIMARY.
- Mostrar información sobre los índices de una tabla.

```
SHOW INDEX FROM pacientes;
```

- Todos los índices pueden ser multicolumna

# ÍNDICES - TIPOS

- 1) "primary key": lo definimos como clave primaria. Los valores indexados deben ser únicos y además no pueden ser nulos. MySQL le da el nombre "PRIMARY". Una tabla solamente puede tener una clave primaria.
- 2) "index": crea un índice común, los valores no necesariamente son únicos y aceptan valores "null". Podemos darle un nombre, si no se lo damos, se coloca uno por defecto. "key" es sinónimo de "index". Puede haber varios por tabla.
- 3) "unique": crea un índice para los cuales los valores deben ser únicos y diferentes. Permite valores nulos y pueden definirse varios por tabla. Podemos darle un nombre, si no se lo damos, se coloca uno por defecto.

**OBJETIVO** → acelerar la recuperación de información.

# ÍNDICES - PRIMARY

- Se crea automáticamente cuando establecemos un campo como clave primaria, no podemos crearlo directamente.
- Los valores indexados deben ser únicos y además no pueden ser nulos.
- Una tabla solamente puede tener una clave primaria por lo tanto, solamente tiene un índice PRIMARY.

```
CREATE TABLE alumnos(  
  matricula varchar(10) not null,  
  documento char(8) not null,  
  apellido varchar(30),  
  nombre varchar(30),  
  domicilio varchar(30),  
  PRIMARY KEY (matricula)  
);
```

```
SHOW INDEX FROM alumnos;
```



# ÍNDICES - INDEX

- Los valores no necesariamente son únicos y aceptan valores "null".
- Puede haber varios por tabla
- En la creación de la tabla, después de la definición de los campos, colocamos "index" seguido del nombre que le damos y entre paréntesis el/los campos por los cuales se indexará dicho índice.

```
CREATE TABLE medicamentos(  
  codigo int unsigned auto_increment,  
  nombre varchar(20) not null,  
  laboratorio varchar(20),  
  precio decimal (6,2) unsigned,  
  cantidad int unsigned,  
  PRIMARY KEY (codigo),  
  INDEX i_laboratorio (laboratorio)  
);
```

```
SHOW INDEX FROM medicamentos;
```

# ÍNDICES - UNIQUE

- los valores deben ser únicos y diferentes (aparece un mensaje de error si intentamos agregar un registro con un valor ya existente).
- Permite valores nulos
- Pueden definirse varios por tabla.

```
CREATE TABLE consultas(  
  fecha date,  
  numero int unsigned,  
  documento char(8) not null,  
  obrasocial varchar(30),  
  medico varchar(30),  
  PRIMARY KEY (fecha,numero),  
  INDEX i_medico (medico),  
  UNIQUE i_consulta (documento,fecha,medico));
```

**SHOW INDEX FROM consultas;**

NO se podrá ingresar una consulta del mismo paciente, en la misma fecha con el mismo médico.

# ÍNDICES - Borrado

```
DROP INDEX i_Prov ON Proveedor;
```

Podemos eliminar los índices creados con "index" y con "unique" pero no el que se crea al definir una clave primaria. Un índice PRIMARY se elimina automáticamente al eliminar la clave primaria:

- Incluir clave primaria: `ALTER TABLE alumno ADD PRIMARY KEY (codigo);`
- Eliminar clave primaria: `ALTER TABLE alumno DROP PRIMARY KEY;`

También funciona:

```
ALTER TABLE Proveedor DROP INDEX i_Prov;
```

# ÍNDICES - Creación (en tabla existente)

Para agregar un índice común a una tabla existente:

```
CREATE INDEX i_peli ON películas (titulo);
```

```
ALTER TABLE películas ADD INDEX i_peli (titulo);
```

Para agregar un índice único a una tabla existente:

```
CREATE UNIQUE INDEX i_tit ON películas (titulo,director);
```

```
ALTER TABLE películas ADD UNIQUE INDEX i_tit (titulo,director);
```

# VISTAS - Intro

Vamos a introducir el concepto de vista para irnos familiarizando con ellas.

- Una vista corresponde a un conjunto de columnas de una o varias tablas.
- Cuando se modifican las tablas originales la vista cambia automáticamente.
- En lenguaje poco técnico podríamos decir que una vista es una consulta usada como tabla

# VISTAS - Creación

```
CREATE VIEW nombre_ciudades_poblacion AS  
  
    SELECT name as nombre_ciudad,population AS poblacion FROM  
    City;  
  
SELECT * FROM nombre_ciudades_poblacion;
```

# IMPORTAR DATOS

El comando **LOAD DATA INFILE** lee registros desde un fichero de texto a una tabla

Sintaxis básica:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Sintaxis más habitual:

```
LOAD DATA INFILE 'Ruta donde tengamos el archivo csv'  
INTO TABLE table donde deseemos almacenar los datos  
FIELDS TERMINATED BY 'string'  
ENCLOSED BY 'Char'  
LINES TERMINATED BY 'string'  
IGNORE number LINES;
```

# IMPORTAR DATOS

Necesitamos tener la tabla creada en MySQL, con los mismos campos que columnas tiene el CSV a importar.

**FIELDS TERMINATED BY ','**

Hace referencia a cómo ha de terminar un campo.

**ENCLOSED BY '""'**

Hace referencia cuando se ha de cerrar (entrecomillar) un valor.

**LINES TERMINATED BY '\n'**

Refiere a cuando la línea del archivo .csv termina con un nuevo carácter.

**IGNORE 1 LINES**

Ignora los encabezados del archivo csv.



# Importar Datos (ejemplo)

1º.- Creamos la tabla “receptora”:

```
mysql > CREATE TABLE Usuarios (  
→ Id INT,  
→ Nombre CHAR(50),  
→ Email VARCHAR(100),  
→ Telefono CHAR(9));
```

2º.- Suponemos que tenemos un archivo .csv del tipo:

```
id;nombre;correo;movil  
1;Antonio;antonio@dominio.com;123456789  
2;Armando;armando@dominio.com;234567891  
3;Carlos;carlos@dominio.com;345678912  
4;Ceferino;cefe@dominio.com;456789123  
5;Cipriano;cipri@dominio.com;567891234
```

3º.- Realizamos el volcado del archivo a la tabla:

```
LOAD DATA INFILE '/tmp/nombres.csv'  
INTO TABLE Usuarios  
FIELDS TERMINATED BY ';'   
LINES TERMINATED BY '\n'  
IGNORE 1 LINES  
;
```

## EJEMPLO PRÁCTICO ([www.generatedata.com](http://www.generatedata.com))

**load data local infile 'C:/Users/buzon/Desktop/BootCamp  
UpgradeHub/MySQL/ROBERTO/Importar bases de datos/datosUsuarios.csv' into table usuarios  
fields terminated by ';' lines terminated by '\n' ignore 1 lines;**

# EXPORTAR TABLAS

1. Mostrar archivos y carpetas ocultas:
  - a. clic derecho en el botón “Inicio”, luego se selecciona “Panel de Control” / “Apariencia y Personalización” / “Opciones del Explorador de **Archivos**” / VER/“**Archivos** y carpetas **ocultas**” a “**Mostrar archivos**, carpetas, y unidades **ocultos**”.
2. 

```
mysql> SELECT * FROM usuarios  
      INTO OUTFILE 'usuariossalida.csv'  
      FIELDS TERMINATED BY '\t' OPTIONALLY ENCLOSED BY "  
      LINES TERMINATED BY '\n';
```
3. Verificar la existencia del archivo en c:\ProgramData\MySQL\MySQL Server 8.0\data\tienda\usuariossalida.csv, en donde ProgramData suele ser una carpeta oculta.

# EXPORTAR TABLAS (no localmente)

1. Detener el servicio de servidor MySQL :
  - a. buscar servicios, y buscar mysql80
  - b. detener servicio
2. Ir a C:\ProgramData\MySQL\MySQL Server 5.6 (ProgramData suele ser carpeta oculta).
3. Abrir el archivo my.ini en el Bloc de notas (posiblemente oculto)
4. Buscar 'secure-file-priv'.
5. Poner el valor a cadena vacía "".
6. Guardar el archivo.
7. Inicie el servicio de servidor MySQL.
8. Abrir CMD como administrador → `icacls c: RutaDestinoDelArchivo /grant everyone:F`
  - a. en Linux `chmod 1777 c: RutaDestinoDelArchivo`

```
mysql> SELECT * FROM usuarios INTO OUTFILE 'usuariosalida.csv'
        FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
        LINES TERMINATED BY '\n';
```

# FIN