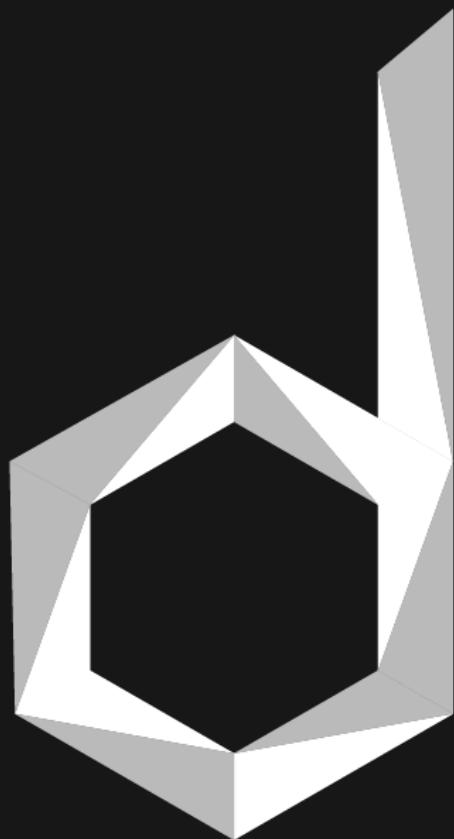


INGENIERÍA MECATRÓNICA



DI\_CERO

DIEGO CERVANTES RODRÍGUEZ

PROGRAMACIÓN: DESARROLLO BACKEND

SQL

Bases de Datos y SQL

# Contenido

<b>Introducción a las Bases de Datos .....</b>	3
Tipos de Bases de Datos.....	3
Representación de las Bases de Datos: Nomenclatura de Chen .....	4
Diagrama ER (Entidad-Relación) .....	4
Diagrama Físico .....	8
Normalización: Tabla de Datos a Base de Datos Relacional (RDB)	10
<b>Ejemplo del Diagrama de una Base de Datos: Blog Posts .....</b>	13
<b>Manejador de Base de Datos Relacional.....</b>	16
Codificación de una Base de Datos en MySQL Workbench .....	19
<b>Base de Datos con Servicios Administrados (Nube) .....</b>	22
Google Cloud Services.....	23
<b>Lenguaje de Programación SQL .....</b>	27
Sub-lenguajes de SQL: <b>DDL (Data Definition Language)</b> .....	27
<b>CREATE:</b> .....	27
Crear una <b>Base de Datos</b> con SQL .....	27
Crear una <b>Tabla</b> con SQL.....	28
Crear una <b>Vista</b> con SQL.....	31
<b>ALTER:</b> .....	31
Alterar una <b>Tabla</b> con SQL.....	31
<b>DROP:</b> .....	33
Borrar una <b>Tabla, Columna o Base de Datos</b> con SQL .....	33
Sub-lenguajes de SQL: <b>DML (Data Manipulation Language)</b> .....	34
<b>INSERT:</b> .....	35
Insertar Datos Nuevos a la <b>Tabla</b> de una <b>Base de Datos</b> con SQL .....	35
<b>UPDATE:</b> .....	36
Editar Datos en la <b>Tabla</b> de una <b>Base de Datos</b> con SQL .....	36
<b>DELETE:</b> .....	37
Borrar Todos los Datos de una Fila Perteneciente a la <b>Tabla</b> de una <b>Base de Datos</b> con SQL .....	37
<b>SELECT:</b> .....	38
Extraer Todos los Datos de una Columna Perteneciente a la <b>Tabla</b> de una <b>Base de Datos</b> .....	38
<b>Ejemplo de Diagrama Físico a Código SQL.....</b>	39

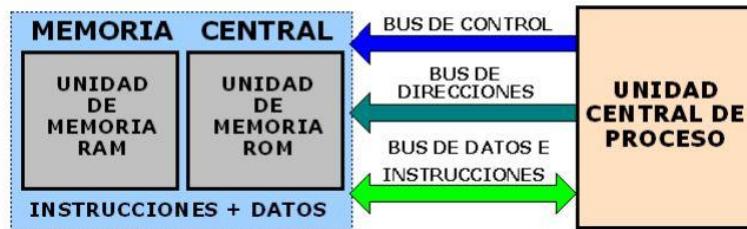
Base de Datos con Código SQL a Diagrama Físico en MySQL Workbench .....	46
<b>Consultas a Bases de Datos: Query - SELECT .....</b>	<b>49</b>
<b>Nested Queries:</b> Consultas Anidadas (Agujero de Conejo) .....	52
Transformar una Pregunta en un Query de SQL - Ejemplos Prácticos .....	54
<b>Bases de Datos No Relacionales .....</b>	<b>59</b>
Jerarquía de Datos de las Bases de Datos No Relacionales: FireStore .....	61
<b>Conclusiones y Aplicaciones .....</b>	<b>62</b>
Big Data .....	62
Data Warehouse .....	62
Data Mining.....	62
ETL.....	62
Business Intelligence.....	63
Machine Learning.....	63
Data Science.....	63
Referencias.....	63



# Introducción a las Bases de Datos

Las bases de datos ayudan a complementar la **arquitectura de Von Neumann**, que es la arquitectura utilizada en ordenadores, la cual a diferencia de la arquitectura Harvard utilizada en microcontroladores, **utiliza una memoria centralizada para realizar sus funciones**. La necesidad de extender la capacidad de la memoria central es la de conservar los datos más allá de la memoria RAM o ROM, ya que en la arquitectura Von Neumann si se contempla el procesamiento de datos, pero no el almacenamiento de datos persistentes, por lo que es de suma importancia la utilización de las bases de datos.

## ARQUITECTURA VON NEUMANN



Para resolver esta situación, donde se busca que de una forma fácil se puedan guardar y extraer datos de información, se obtuvieron dos soluciones:

- **Bases de datos basadas en archivos**: Este método de almacenamiento de datos persistentes consiste en guardar información en un archivo de texto plano, hojas de cálculo, etc. usualmente separados por comas o de alguna otra forma ordenada.
- **Bases de datos basadas en documentos**: En este tipo de base de datos, la unidad básica de almacenamiento es el documento, que puede contener datos en forma de texto, números, listas, objetos JSON (JavaScript Object Notation) y a veces incluso otros documentos anidados.

## Tipos de Bases de Datos

Los diferentes tipos de bases de datos existentes son los siguientes:

- **Relacionales o RDB**: Son bases de datos basadas en documentos que se rigen por las 12 reglas de Edgar Codd, que dan como resultado el álgebra relacional, a través de las cuales se indican las reglas con las que los datos de las RDB se pueden mezclar o relacionar entre sí.
  - **Privadas**: Microsoft SQL Server, Oracle, etc.
  - **Open Source**: PostgreSQL, MySQL, MariaDB, etc.

### Ejemplos de bases de datos relacionales



- **No relacionales o NRDB:** Hay varios tipos de bases de datos no relacionales, todas ellas no se conforman de un solo tipo de bases de datos, sino de varios, y aunque puedan ser muy distintas unas de otras, todas se engloban dentro de la misma categoría de base de datos no relacional que utilizan lenguajes NoSQL (Not Only SQL). Los diferentes tipos de bases de datos no relacionales son:
  - Basadas en Clave-Valor, en Documentos, en Grafos, en Memoria, Optimizadas para Búsquedas, etc. algunos ejemplos de ellas son:
    - Memcached, Cassandra (Facebook), DynamoDB, ElasticSearch, BigQuery, Neo4j (GraphQL), MongoDB, Firestore (Firebase).

## Bases de datos no relacionales



- **Auto Administradas:** En este tipo de bases de datos se instala, actualiza y mantiene el software en un ordenador de forma local y la consistencia de datos se realiza de forma manual.
- **Administradas:** Este tipo de base de datos se ofrece por las nubes modernas como las proporcionadas por Amazon, Google, Azure (Microsoft), para ello la instalación no se realiza de forma local y, por lo tanto, no se mantiene la consistencia de datos de forma manual, sino que se realiza de forma automática por el servicio de la nube.

## Representación de las Bases de Datos: Nomenclatura de Chen

### Diagrama ER (Entidad-Relación)

- **Entidad:** Una entidad es algo muy similar a un objeto, el cual se puede asociar con ciertos **atributos (características)**, de la misma forma como se maneja en POO.
  - **Atributo:** Se representa por medio de un **óvalo simple** cuando la entidad solo posee **uno solo de ese atributo**, si cuenta con más de uno, esto se indica con **dos óvalos anidados** que rodeen el nombre del atributo, a esto se le llama **atributo multivalor**.
    - **Ejemplo 1:** Cualquier automóvil posee un solo volante, pero varias llantas, por lo cual el atributo “volante” será rodeado por un óvalo simple y el atributo “llantas” se rodeará de un óvalo doble.
    - **Ejemplo 2:** Ahora se representará a través de un diagrama de Chen las entidades (objetos) laptops, donde cabe mencionar que **los atributos donde se subraye su nombre** se llaman **atributos clave** y diferencian cada laptop individualmente (instancia), además los atributos que tengan un **óvalo con línea punteada** representan los **atributos derivados**, que corresponden a datos que se pueden obtener a través de otros y además que los atributos pueden tener otros atributos relacionados.

## Entidades



## Atributos

no de serie	color	año	pantalla
LKJ789JKAS	gris	2017	AX4829i
KCO3100KJH	negro	2019	AX4930i
NSDJOIH128	negro	2018	AX4930i
09KSIHBD71	gris	2017	AX4829i

- **Tipos de atributos clave:** Los atributos clave pueden ser naturales, esto significa que son pertenecientes al objeto y no se pueden remover y los atributos clave artificiales, que se asignan de manera arbitraria.
- **Entidades fuertes:** No dependen de otra entidad para existir, estas se rodean de un cuadrado simple.
  - **Entidades débiles:** Si dependen de otra entidad para existir, estas se rodean de un cuadrado doble, así como los atributos multivalor. Además, existen dos tipos de debilidad:
    - **Debilidad por identidad:** Que para que se puedan diferenciar, deben tomar el atributo clave de la entidad de la que dependen.
    - **Debilidad por existencia:** Que pueden tener un identificador propio, pero aun así dependen de otra identidad para existir.

## Entidades débiles



## Entidades débiles: identidad

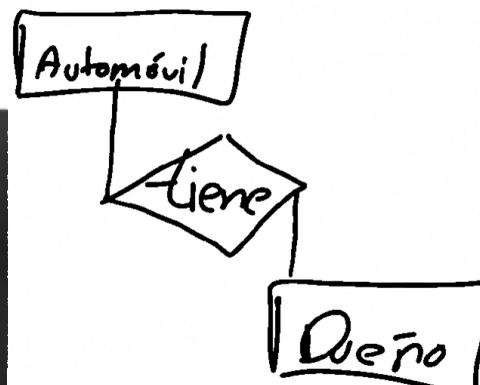
Libros			Ejemplares		
id	título	...	libro_id	localización	edición
LKJ789JKAS	Viaje al cent...	...	LKJ789JKAS	pasillo 1	1
KCO3100KJH	El señor de ...	...	KCO3100KJH	pasillo 1	1
NSDJOIH128	De la tierra...	...	NSDJOIH128	pasillo 1	3
09KSIHBD71	Amor en tie...	...	09KSIHBD71	pasillo 1	1

## Entidades débiles: existencia

Libros			Ejemplares		
id	título	...	id	localización	edición
LKJ789JKAS	Viaje al cent...	...	JKE7823CLK	pasillo 1	1
KCO3100KJH	El señor de ...	...	JKFE1093JD	pasillo 1	1
NSDJOIH128	De la tierra...	...	82938ISHDIK	pasillo 1	3
09KSIHBD71	Amor en tie...	...	838439JHDUI	pasillo 1	1

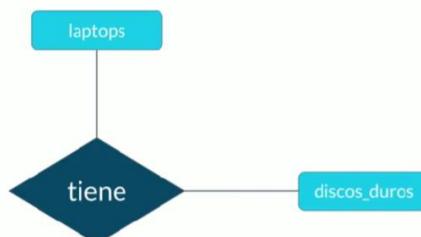
- **Relación:** Es la conexión con las que se ligan las diferentes entidades entre sí, para ello dentro de las relaciones se utilizan verbos que conecten una relación con la otra.

## Relaciones



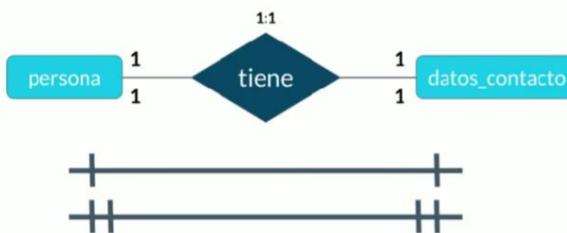
- **Cardinalidad:** Una peculiaridad de las relaciones es que a través de ellas se deben separar los atributos multivaluados, ya que cada uno puede tener características específicas y se relacionan con el concepto de cardinalidad porque este se relaciona con el número de veces que se repite un atributo en una entidad.
  - Cuando se utiliza la cardinalidad se utiliza todo el concepto del verbo de conexión que utiliza la relación para indicar cual es el número de entidades con las que cuenta una entidad, de esta manera es como se puede separar los atributos multivaluados.

## Relaciones

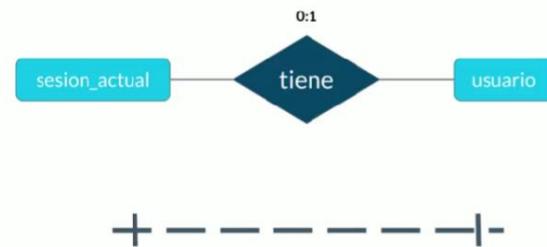


- De igual manera se obtiene la cardinalidad de ambos lados, ya que esto puede variar cuando se ve desde perspectivas distintas. Además, se maneja cierta nomenclatura en el diagrama para denotarlo.

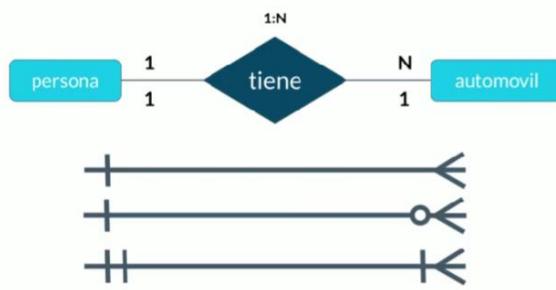
### Cardinalidad: 1 a 1



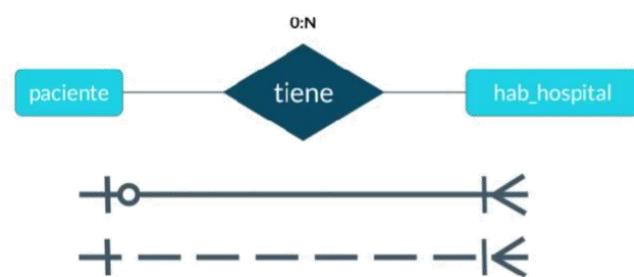
### Cardinalidad: 0 a 1



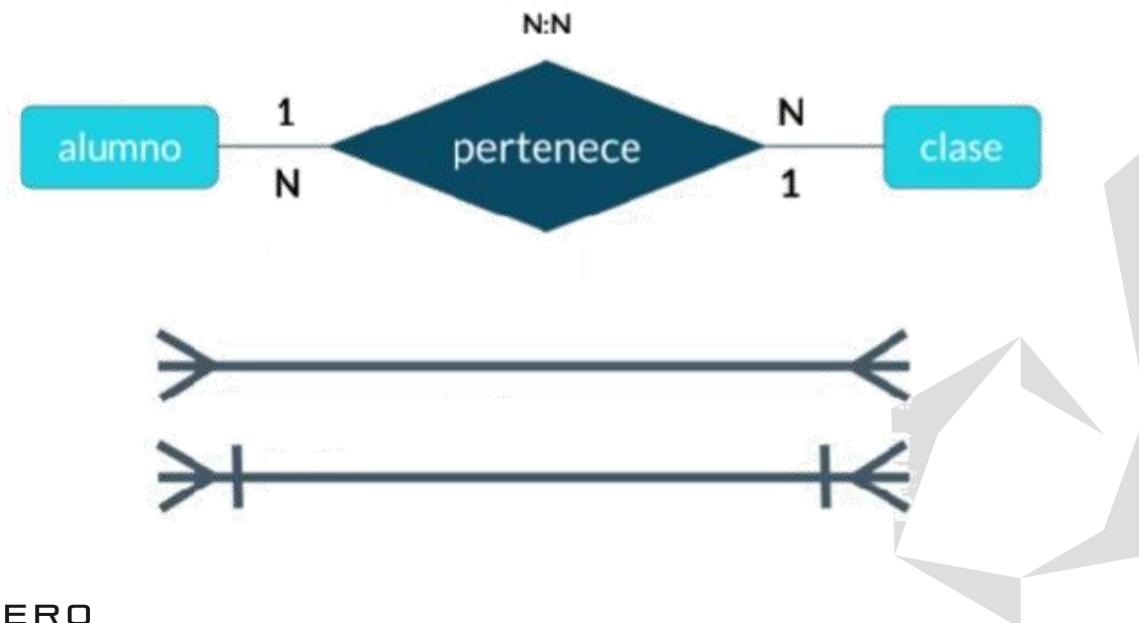
### Cardinalidad: 1 a N



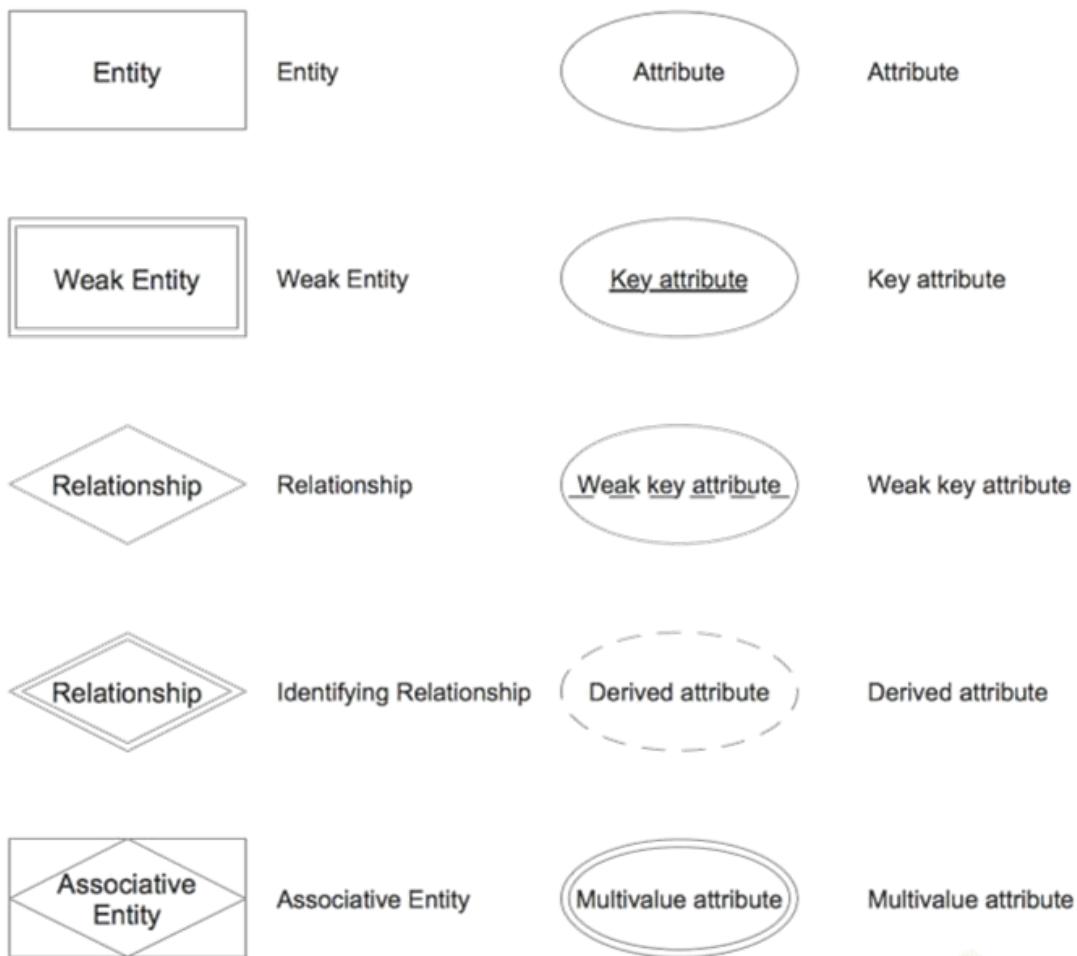
### Cardinalidad: 0 a N



### Cardinalidad: N a N



Todos los conceptos explicados previamente que describen los datos almacenados en una base de datos relacional se deben plasmar en un diagrama ER (entidad-relación), para ello se utiliza la siguiente simbología:



### Diagrama Físico

Además del **diagrama ER (entidad-relación)** existe otro llamado **diagrama físico** (que se deriva del ER), el cual es más específico ya que **menciona los tipos de datos**, que pueden ser los siguientes:

- **Texto:**
  - **Char(n):** Minimiza el espacio de memoria a solo los caracteres que ocupa el texto.
  - **VarChar(n):** Utiliza el espacio de memoria de forma dinámica, reservando como mínimo un espacio de memoria y extendiéndolo si es necesario hasta 255 caracteres.<sup>3</sup>
  - **Text:** Reserva el espacio de memoria para cadenas de caracteres (palabras u oraciones) muy grandes.
- **Números:**
  - **Enteros:** Integer, BigInt y SmallInt.
  - **Decimales:** Decimal(n, s) y Numeric(n,s ), donde n es el número y s indica cuantos decimales aparecen de dicho número.

- **Fecha/Hora:**
  - **Date:** Contiene año, fecha y día.
  - **Time:** Contiene solo la hora.
  - **Datetime y Timestamp:** Contienen la fecha y la hora.
- **Lógicos:**
  - **Boolean:** Puede adoptar valores true (1) o false (0).

## Tipos de dato

Texto	Números	Fecha/hora	Lógicos
CHAR(n)	INTEGER	DATE	BOOLEAN
VARCHAR(n)	BIGINT	TIME	
TEXT	SMALLINT	DATETIME	
	DECIMAL(n, s)	TIMESTAMP	
	NUMERIC (n, s)		

Además del tipo de dato, se indican las restricciones (reglas) de la base de datos que delimitan el tipo de dato que admite, cuántos datos admite, etc.

## Constraints (Restricciones)

Constraint	Descripción
NOT NULL	Se asegura que la columna no tenga valores nulos
UNIQUE	Se asegura que cada valor en la columna no se repita
PRIMARY KEY	Es una combinación de NOT NULL y UNIQUE
FOREIGN KEY	Identifica de manera única una tupla en otra tabla
CHECK	Se asegura que el valor en la columna cumpla una condición dada
DEFAULT	Coloca un valor por defecto cuando no hay un valor especificado
INDEX	Se crea por columna para permitir búsquedas más rápidas

- **Índice:** Es un elemento cuya ventaja es que permite realizar búsquedas de datos en la columna de una tabla de una base de datos, pero la desventaja que tiene es que hace lento el procesamiento de datos en esa columna. Por lo que su mayor utilidad es cuando en una base de datos se estarán realizando consultas constantes, pero no se introducirán datos nuevos de forma continua.

## Normalización: Tabla de Datos a Base de Datos Relacional (RDB)

El proceso de normalización permite obtener una base de datos a partir de una tabla de datos, separándolas en los componentes previamente explicados, como lo son las entidades, atributos, etc., para ello se aplican las 12 reglas del álgebra relacional de Codd, también llamadas formas normales o FN, que establecen la base de datos como relacional.

## Normalización



A continuación, se denotará este concepto con un ejemplo, donde partiendo de una tabla de datos, estos se organizarán para ser normalizados:

## Sin normalizar

alumno	nivel_curso	nombre_curso	materia_1	materia_2
Juanito	Maestría	Data engineering	MySQL	Python
Pepito	Licenciatura	Programación	MySQL	Python

Las formas normales que se siguen para normalizar la tabla son las siguientes:)

- **1FN (Primera Forma Normal) - Atributos atómicos:** Esta norma indica que no se pueden tener campos repetidos y sus atributos deben ser atómicos. Un atributo es atómico si los elementos del dominio son simples e indivisibles.

alumno	nivel_curso	nombre_curso	materia_1	materia_2
--------	-------------	--------------	-----------	-----------

Juanito	Maestría	Data engineering	MySQL	Python
---------	----------	------------------	-------	--------

Pepito	Licenciatura	Programación	MySQL	Python
--------	--------------	--------------	-------	--------

alumnos				
alumno_id	alumno	nivel_curso	nombre_curso	materia
1	Juanito	Maestría	Data engineering	MySQL
1	Juanito	Maestría	Data engineering	Python
2	Pepito	Licenciatura	Programación	MySQL
2	Pepito	Licenciatura	Programación	Python

- **2FN (Segunda Forma Normal) - Clave Única:** Esta norma indica que cada campo de la tabla debe depender de una clave única, si no es posible, se debe separar en entidades distintas.

alumnos				
alumno_id	alumno	nivel_curso	nombre_curso	materia
1	Juanito	Maestría	Data engineering	MySQL
1	Juanito	Maestría	Data engineering	Python
2	Pepito	Licenciatura	Programación	MySQL
2	Pepito	Licenciatura	Programación	Python

alumnos			
alumno_id	alumno	nivel_curso	nombre_curso
1	Juanito	Maestría	Data engineering
2	Pepito	Licenciatura	Programación

materias		
materia_id	alumno_id	materia
1	1	MySQL
2	1	Python
3	2	MySQL
4	2	Python



- **3FN (Tercera Forma Normal) - Campos Clave Sin Dependencias:** Esta norma indica que los campos clave no deben tener dependencias, osea que aquellos datos que no pertenecen a la entidad deben tener una independencia de las demás y un campo clave propio.

alumnos			
alumno_id	alumno	nivel_curso	nombre_curso
1	Juanito	Maestría	Data engineering
2	Pepito	Licenciatura	Programación

materias		
materia_id	alumno_id	materia
1	1	MySQL
2	1	Python
3	2	MySQL
4	2	Python

alumnos		
alumno_id	alumno	curso_id
1	Juanito	1
2	Pepito	2

cursos		
curso_id	nivel_curso	nombre_curso
1	Maestría	Data engineering
2	Licenciatura	Programación

materias		
materia_id	alumno_id	materia
1	1	MySQL
2	1	Python
3	2	MySQL
4	2	Python

- **4FN (Cuarta Forma Normal) - Campos Multivaluados:** Esta norma indica que los campos multivaluados deben ser identificados y separados por una clave única, evitando así que se repitan en cada entidad.

alumnos		
alumno_id	alumno	curso_id
1	Juanito	1
2	Pepito	2

cursos		
curso_id	nivel_curso	nombre_curso
1	Maestría	Data engineering
2	Licenciatura	Programación

materias	
materia_id	materia
1	MySQL
2	Python

materias_por_alumno		
mpa_id	materia_id	alumno_id
1	1	1
2	2	1
3	1	2
4	2	2

## Ejemplo del Diagrama de una Base de Datos: Blog Posts

Para crear una base de datos, primero que nada, debemos pensar en las **entidades** que se utilizarán en ella y posteriormente deberemos pensar en los **atributos** que le pertenecen, todo esto se coloca en un **diagrama ER (entidad-relación)** para modelar cómo los datos se relacionan entre sí. En el caso del blog post son los siguientes:

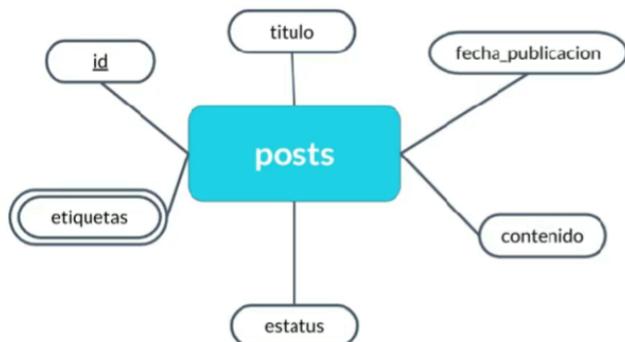
- **Entidades:**
  - Posts (Publicaciones).
  - Usuarios.
  - Comentarios.
  - Categorías.
  - Etiquetas.

### Diagrama ER: Platziblog



- **Atributos de las Entidades:**
  - **Atributos Entidad Posts:**
    - Título.
    - Fecha\_publicacion.
    - Contenido.
    - Estatus (Check Activo o Inactivo).
    - Etiquetas (Categoría interna).
    - Id (Clave única) (Primary Key o PK).

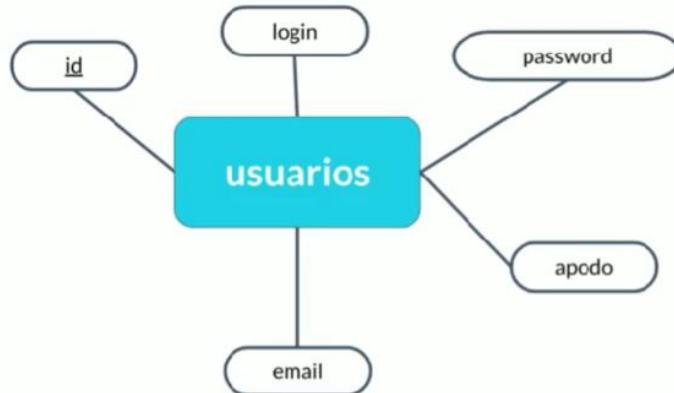
### Entidades Platziblog



- **Atributos Entidad Usuarios:**

- Login (Nombre de Usuario) (**Not Null o NN**).
- Password (**Not Null o NN**).
- Nickname (**Not Null o NN**).
- Email (**Not Null o NN y Unique**).
- Id (**Primary Key o PK**).

## Entidades Platziblog

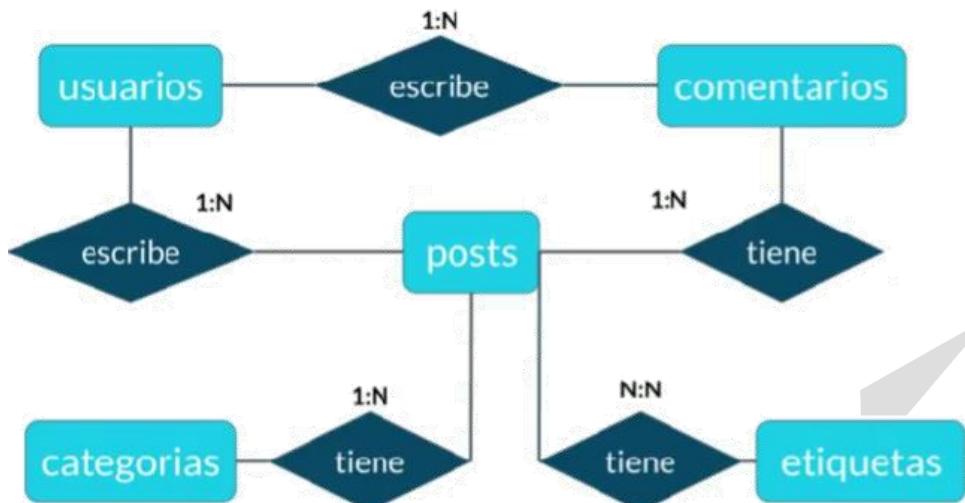


- **Diagrama ER (Entidad-Relación):**

- **Relaciones y Cardinalidad:**

- Un usuario tiene (puede escribir) varios posts.
- Un usuario tiene (puede escribir) varios comentarios.
- Un post tiene varios comentarios.
- Una categoría tiene (engloba) varios posts.
- Un post tiene varias etiquetas y una etiqueta tiene (engloba) varios posts.

## Diagrama ER: Platziblog



- **Diagrama Físico (Tipo de Dato y Constraints):**

- **Tipos de Datos:**

Texto	Números	Fecha/hora	Lógicos
CHAR(n)	INTEGER	DATE	BOOLEAN
VARCHAR(n)	BIGINT	TIME	
TEXT	SMALLINT	DATETIME	
	DECIMAL(n, s)	TIMESTAMP	
	NUMERIC (n, s)		

- **Constraints (Restricciones):**

- **PK: Primary Key.**

- Esta clave debe estar ligada con una foreign key de alguna entidad que sea dependiente de ella.

- **FK: Foreign Key.**

- Esta clave debe estar ligada con una primary key para ver de qué entidad depende, para ello nos debemos fijar en la cardinalidad del diagrama.

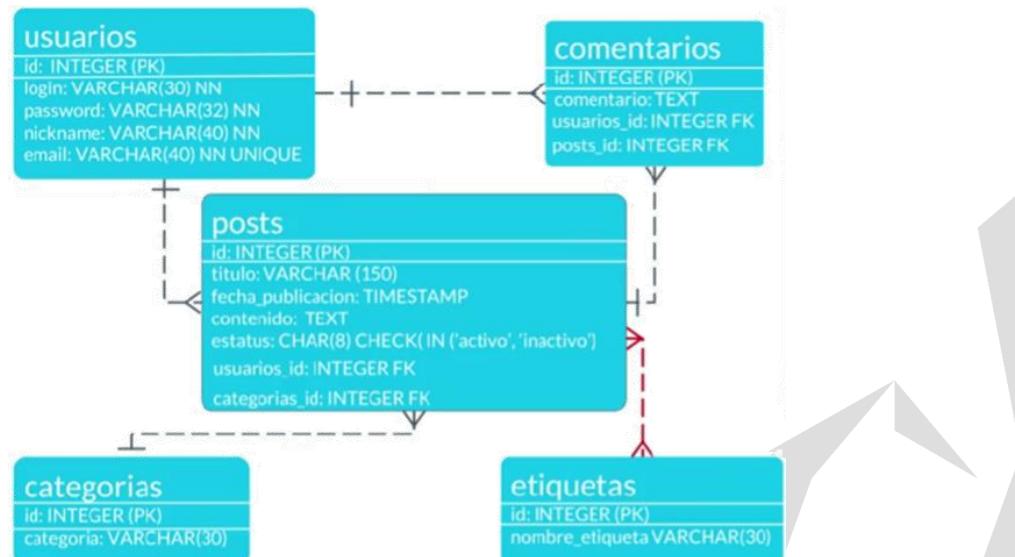
- **NN: Not Null.**

- **UNIQUE.**

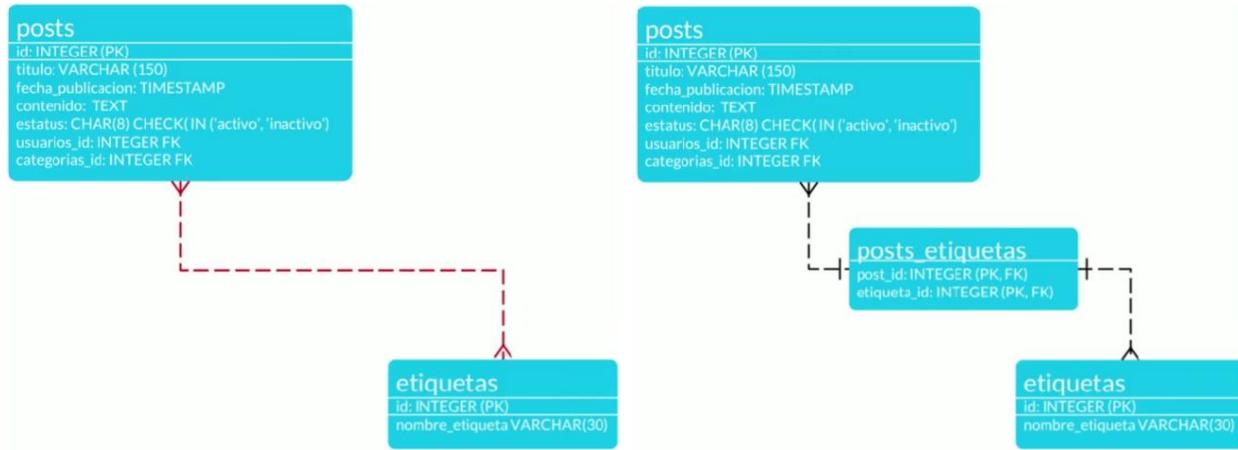
- **CHECK.**

Constraint	Descripción
NOT NULL	Se asegura que la columna no tenga valores nulos
UNIQUE	Se asegura que cada valor en la columna no se repita
PRIMARY KEY	Es una combinación de NOT NULL y UNIQUE
FOREIGN KEY	Identifica de manera única una tupla en otra tabla
CHECK	Se asegura que el valor en la columna cumpla una condición dada
DEFAULT	Coloca un valor por defecto cuando no hay un valor especificado
INDEX	Se crea por columna para permitir búsquedas más rápidas

- **Diagrama relacional global sin cardinalidad N:N:**



- **Diagrama N:N Intermedio o de Pivote:** Este se utiliza cuando se tiene una relación con cardinalidad de N:N entre dos entidades, para ello se debe agregar un diagrama intermedio que relacione ambos id. Es importante mencionar que para crear las llaves únicas de estas relaciones se deben combinar ambos id.



## Manejador de Base de Datos Relacional

Para poder experimentar con bases de datos se puede instalar un manejador de bases de datos relacionales (RDBMS o Relational Data Base Manager System) en nuestro sistema operativo Windows, aunque ya que se maneja una base de datos en producción, normalmente se utilizan servicios de nube.

MySQL es de los manejadores open source más populares del mercado, razón por la cual es el elegido para utilizarse, para ello debemos descargar la versión del instalador 5.7, que se encuentra disponible en el siguiente enlace:

<https://dev.mysql.com/downloads/windows/installer/5.6.html>

### ④ MySQL Community Downloads

◀ MySQL Installer

MySQL Installer 5.7.44

**Note:** MySQL 8.0 is the final series with MySQL Installer. As of MySQL 8.1, use a MySQL product's MSI or Zip archive for installation. MySQL Server 8.1 and higher also bundle MySQL Configurator, a tool that helps configure MySQL Server.

Select Version: 5.7.44

Select Operating System: Microsoft Windows

Windows (x86, 32-bit), MSI Installer	5.7.44	2.1M	Download
(mysql-installer-web-community-5.7.44.0.msi)	MDS: 6cc27e2a42a54b593a9d3544f2529a53   Signature		

Windows (x86, 32-bit), MSI Installer	5.7.44	373.7M	Download
(mysql-installer-community-5.7.44.0.msi)	MDS: e89af3ba9b4716ff5e647b0fd2edab2   Signature		

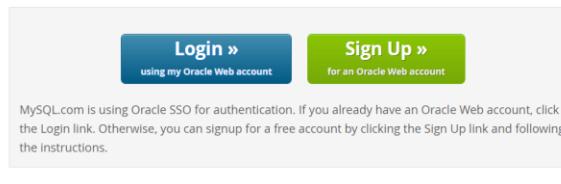
Después daremos clic en el botón de seguir con la descarga, o si queremos también nos podemos registrar a la plataforma de MySQL.

## MySQL Community Downloads

Login Now or Sign Up for a free account.

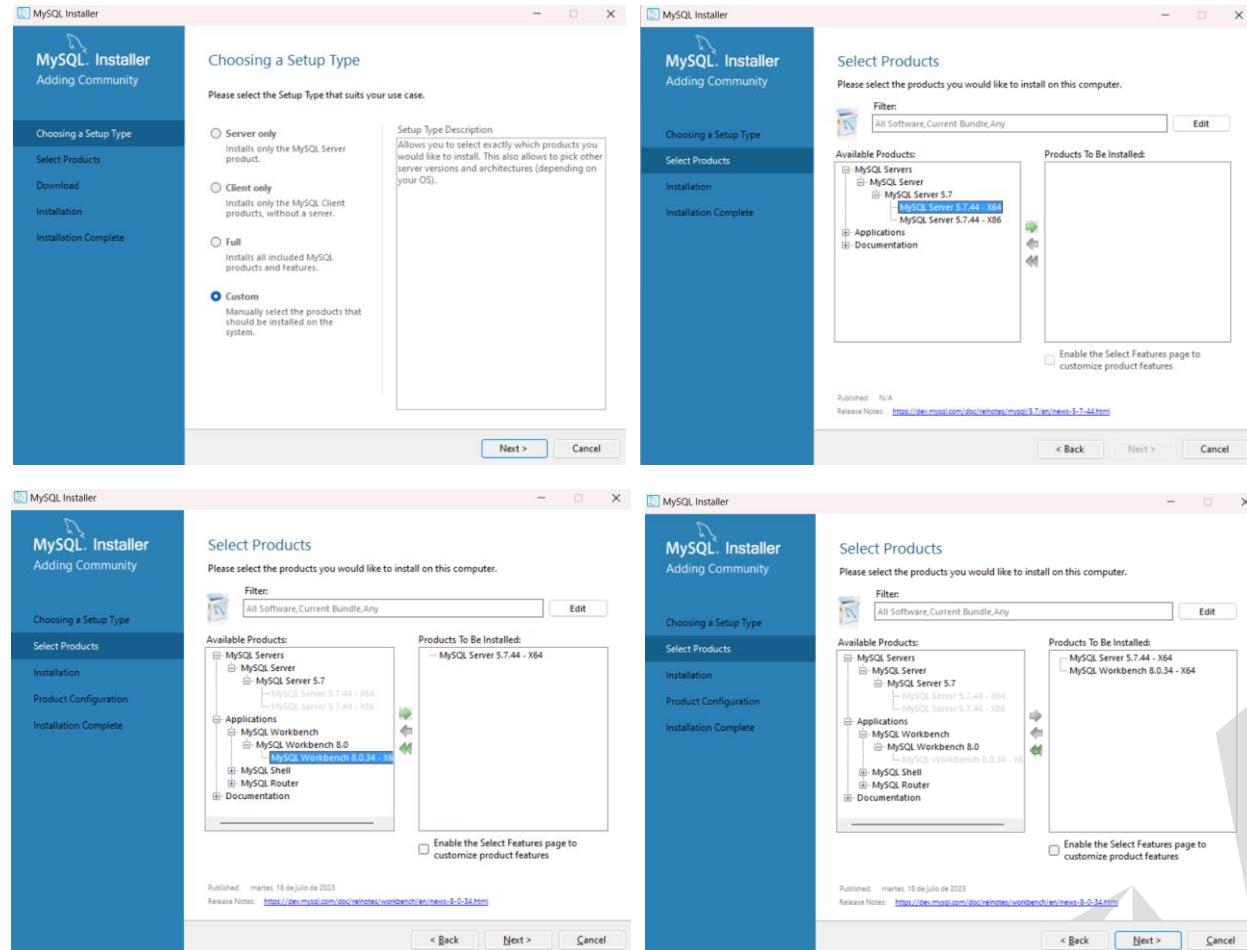
An Oracle Web Account provides you with the following advantages:

- 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

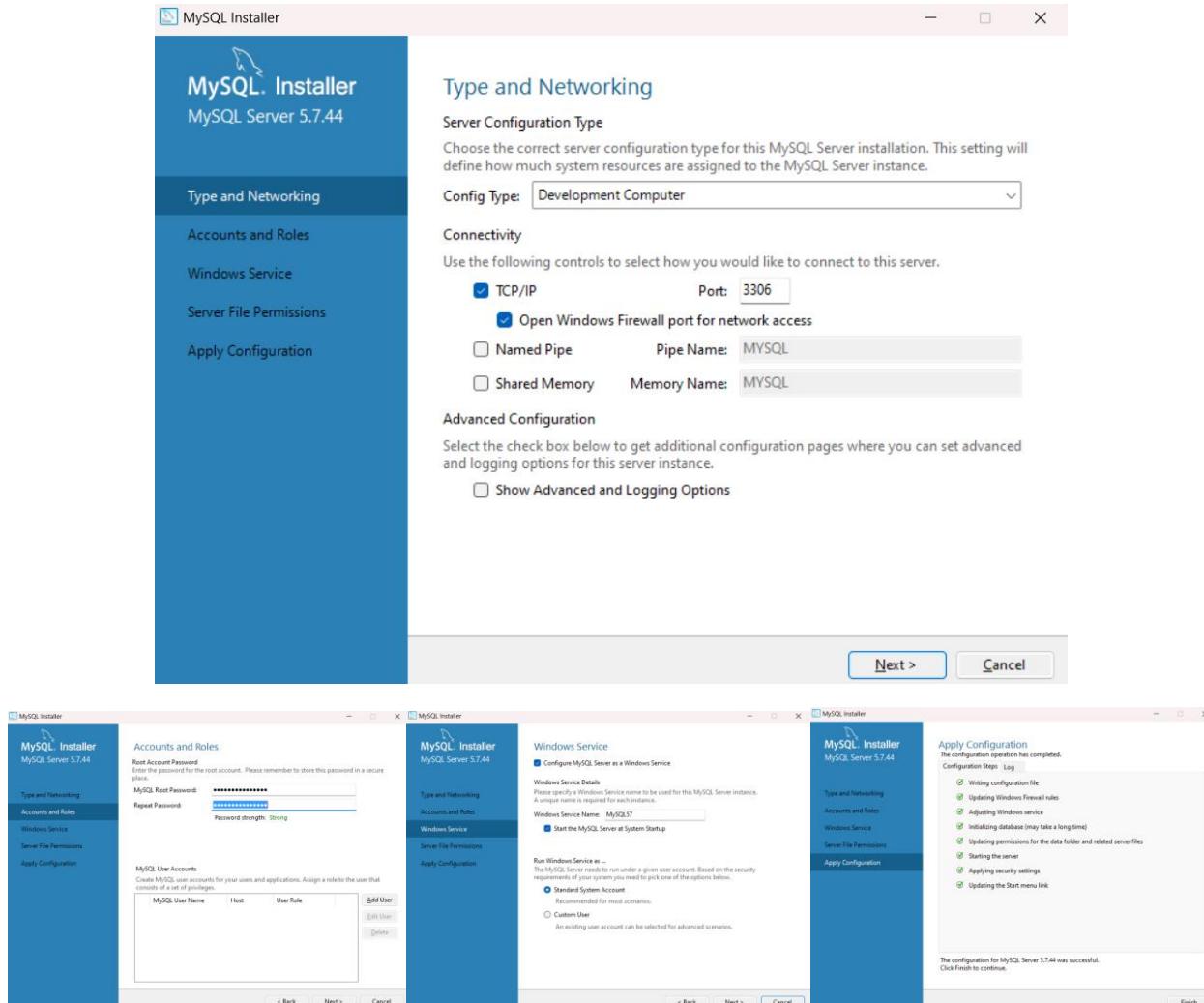


[No thanks, just start my download.](#)

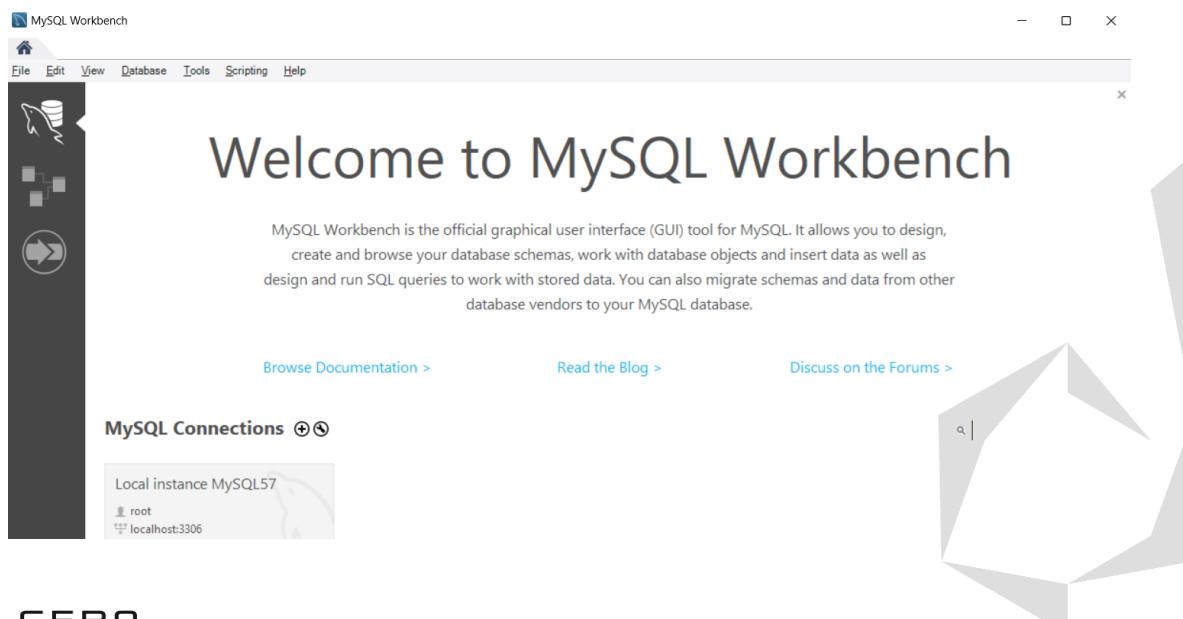
Luego daremos clic en la instalación Custom → MySQL Servers → MySQL Server 5.7 → MySQL Server 5.7 - X64 → → → Applications → MySQL Workbench → MySQL Workbench → Next → Execute.



Config Type → Development Computer → Port → Puerto Deseado, este lo deberemos recordar → Next → **MySQL Root Password:** Contraseña del usuario principal de la DB, tiene todos los permisos → Execute.

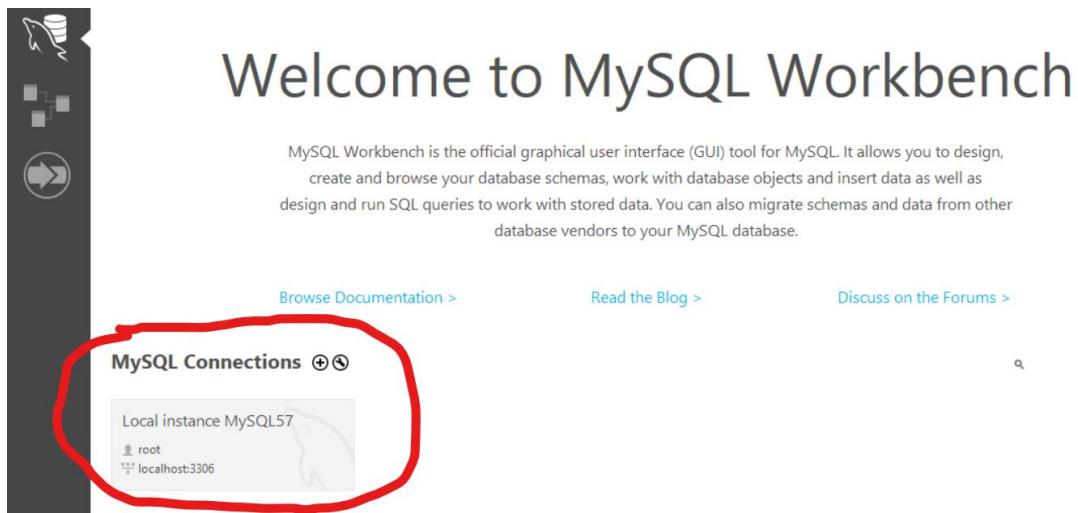


Al finalizar este procedimiento ya estaré instalado el sistema de manejo de la base de datos RDBMS llamado MySQL Workbench, el cual es llamado cliente gráfico y permite ver cómo funciona la base de datos internamente en forma de tablas.

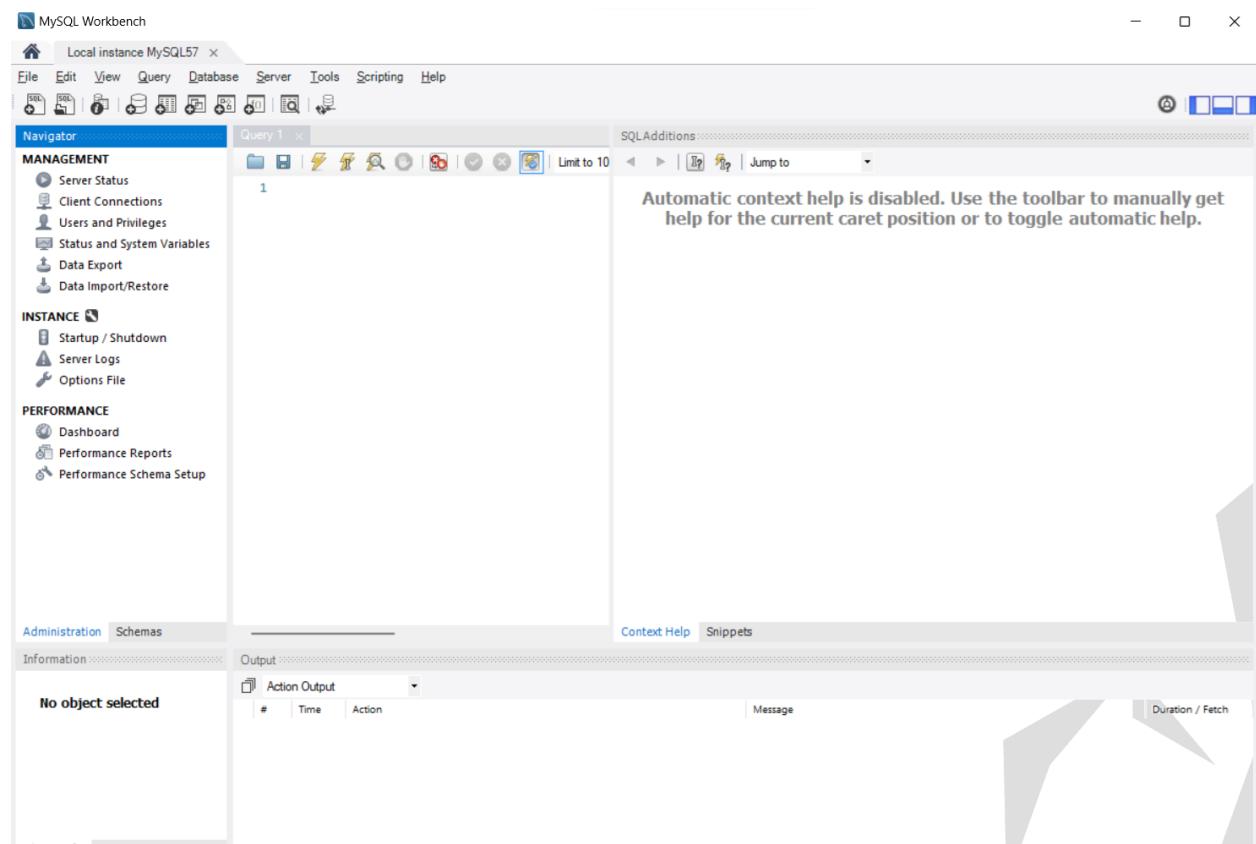


## Codificación de una Base de Datos en MySQL Workbench

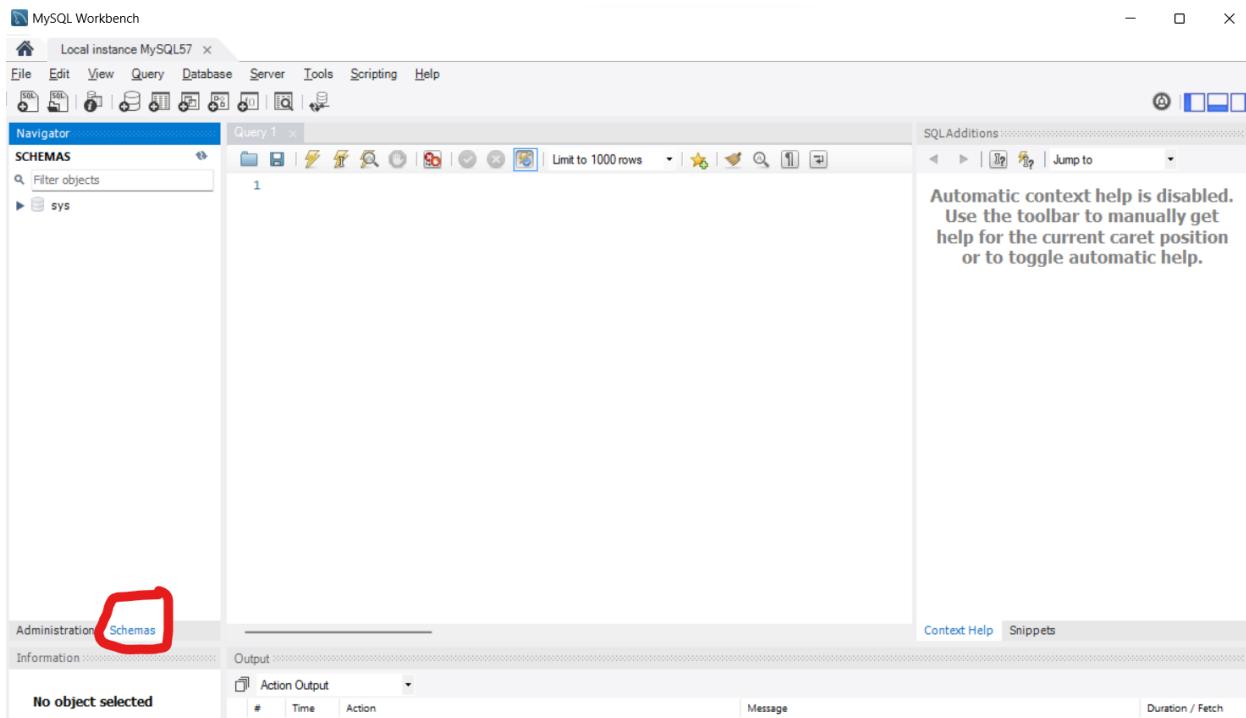
En la pantalla principal del RDBMS MySQL Workbench se puede observar una parte que dice MySQL Connections, en esta se muestran todas las bases de datos que se pueden manejar con este cliente gráfico, indicando su nombre, nombre de usuario (root) y puerto. En esta misma parte si se quisiera agregar una base de datos adicional, se podría realizar al dar clic en el símbolo de +.



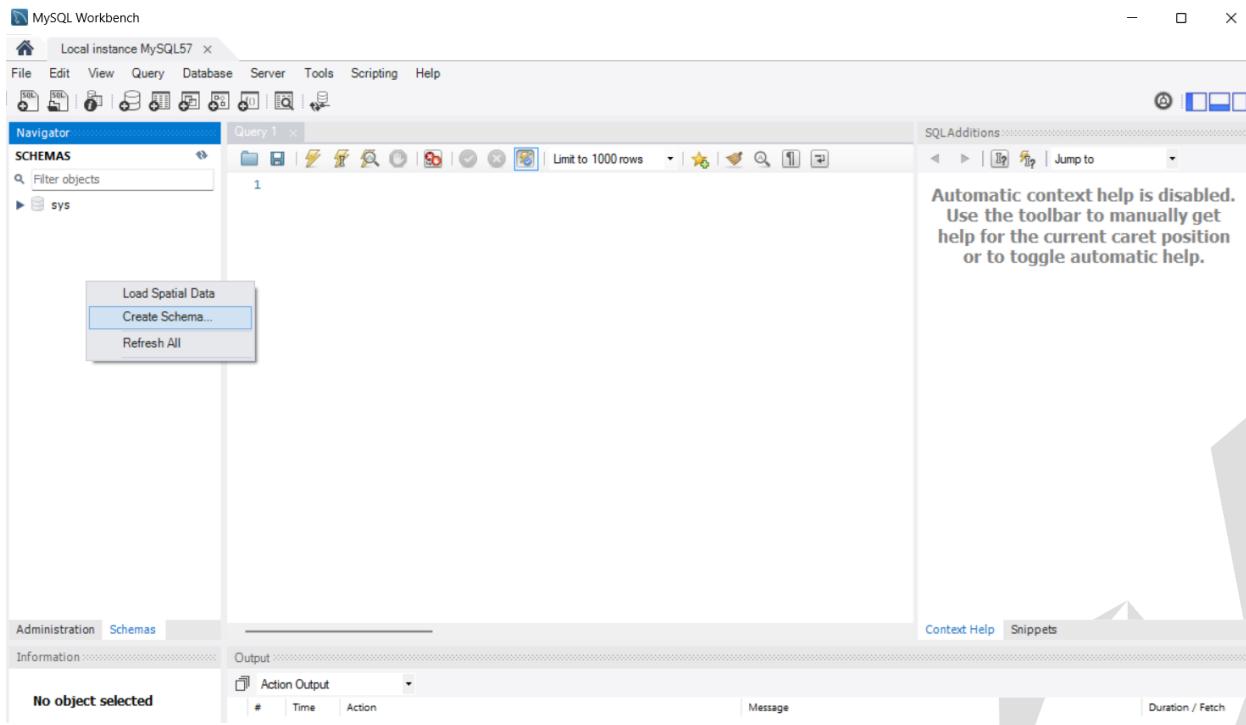
Al dar clic en estas conexiones a las bases de datos, me pedirá la contraseña de acceso y después de dársela podremos ya manejar la base de datos a través del lenguaje SQL.



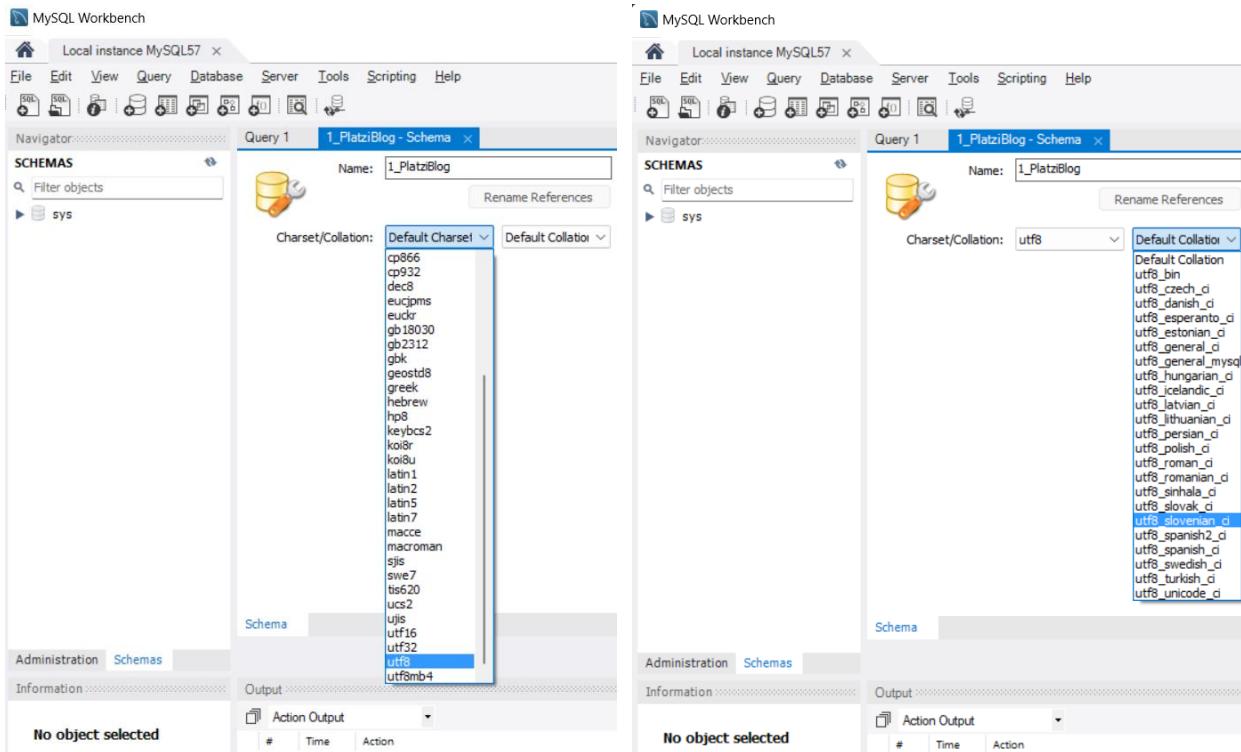
Para acceder a la información de las bases de datos, debemos dar clic en la pestaña de Schemas, ya que esta es una forma alterna de referirnos a ellas:



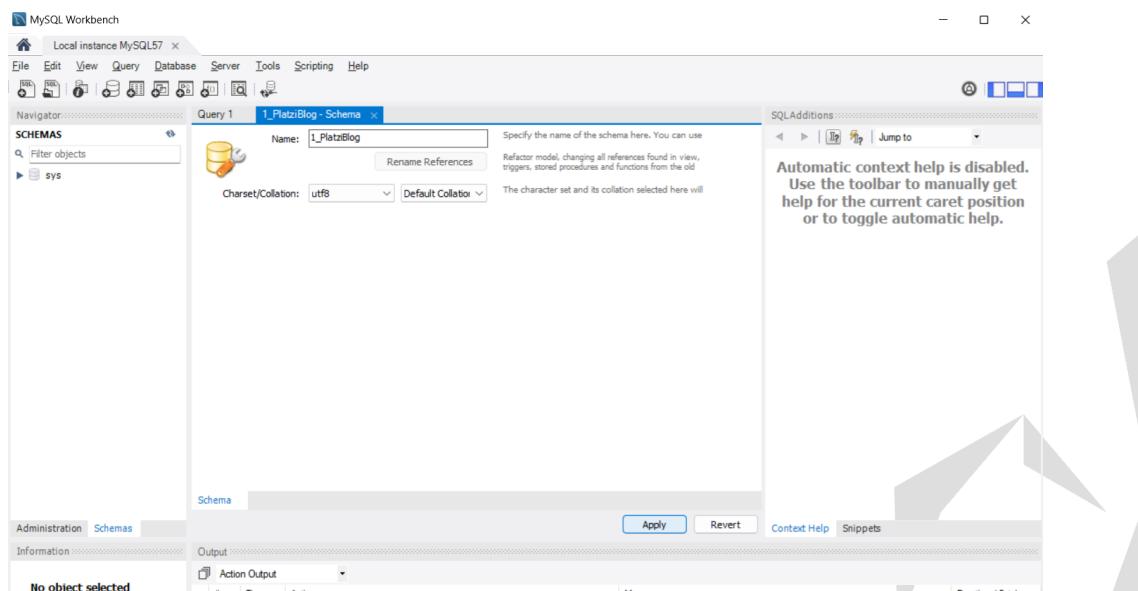
Para crear una base de datos nueva, daremos clic derecho sobre el área de trabajo de los Schemas y seleccionaremos la opción de Create Schema...



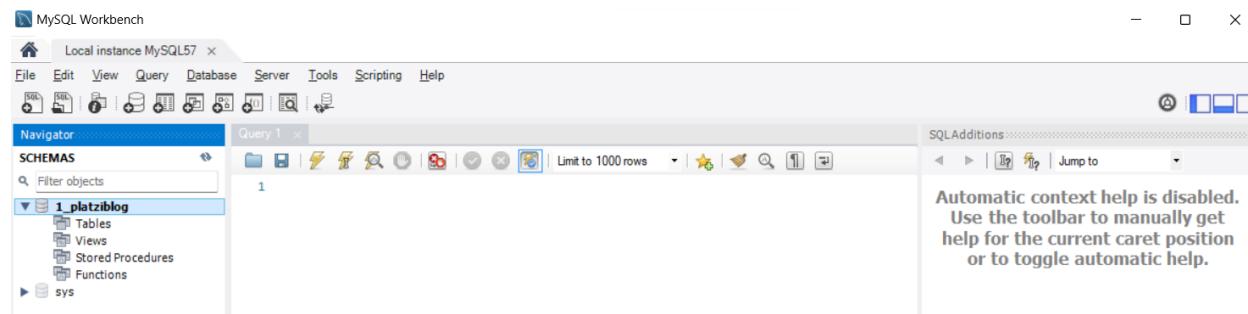
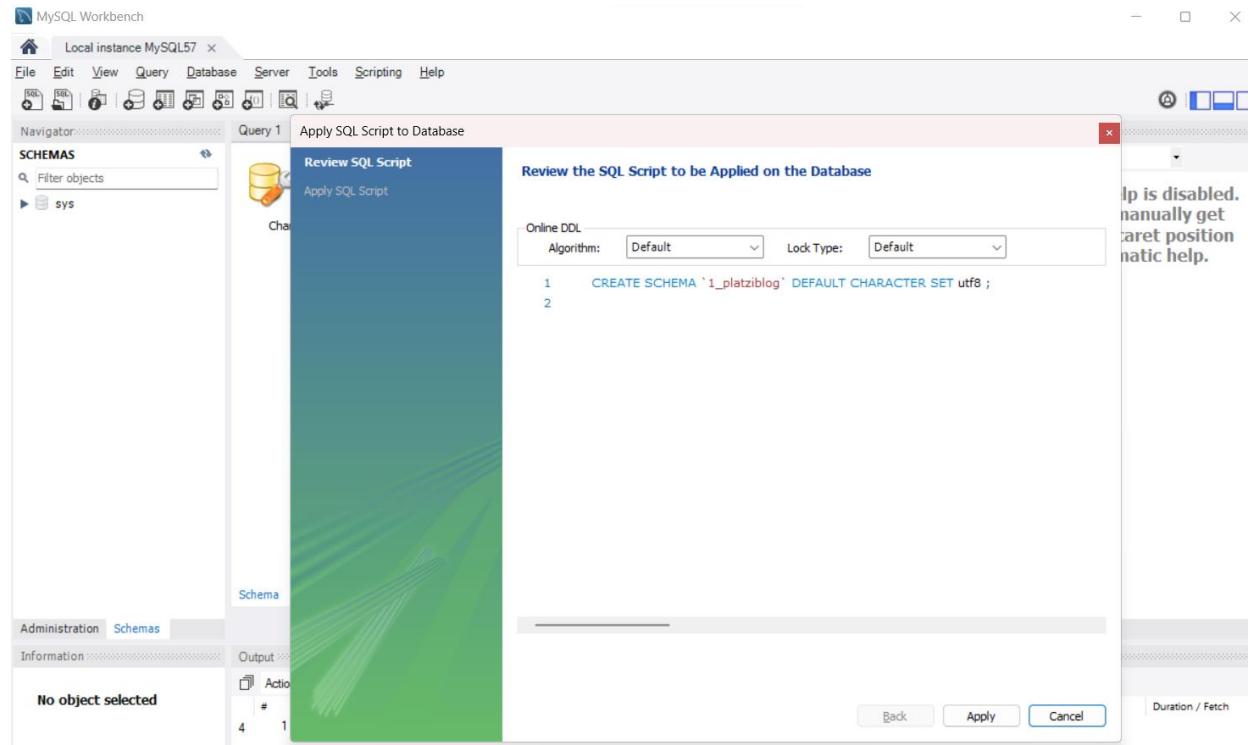
Luego asignaremos un nombre para la base de datos y seleccionaremos la opción de Charset/Collation para indicar la declaración de codificación o codificación de caracteres, donde al especificar la opción utf-8 (caracteres Unicode) como la codificación, nos aseguramos de que el archivo pueda contener caracteres especiales, letras acentuadas y otros caracteres no ASCII sin problemas (recordemos que esta misma codificación la indicábamos al inicio de los scripts creados en el lenguaje de programación Python), además podríamos hasta especificar el idioma que se utilizará en la base de datos, pero en este caso eso no se indicará.



Ya que se haya especificado la codificación, se dará clic en el botón de Apply.



Al haber dado clic en el botón de Apply, aparecerá una ventana donde se indica la configuración indicada a través de lenguaje SQL, a esta ventana se le da clic de nuevo en el botón de Apply → Finish y al hacerlo aparecerá una nueva pestaña en el área de trabajo de SCHEMAS con el nombre indicado.



## Base de Datos con Servicios Administrados (Nube)

Muchas empresas no utilizan un manejador de bases de datos relacionales local (RDBMS o Relational Data Base Manager System) para gestionar sus bases de datos, sino que utilizan algo llamado Servicio Administrado o Cloud, el cual se encuentra en la nube y por ello no se debe administrar de forma manual la base de datos en cuestiones de seguridad, actualizaciones, crecimiento, sistema operativo, redes, etc. solo se centran en su funcionalidad, pero no en su hardware o mantenimiento. Algunos ejemplos muy famosos de ello son AWS, Google Cloud Platform, Azure, etc. pero su desventaja es que la mayoría de ellos son servicios de pago, no open source para desarrolladores, razón por la cual están más dirigidos a empresas

## Google Cloud Services

Para este ejemplo se utilizará el periodo de prueba de Google Cloud, el cual se encuentra en el siguiente enlace, donde al ingresar me llevará por default al último proyecto de Google que haya creado, pero como este será un proyecto nuevo, deberé dar clic en el desplegable que tiene el nombre del proyecto y dar clic en el botón de PROYECTO NUEVO:

<https://console.cloud.google.com/>

The screenshot shows the Google Cloud Console interface. At the top, there's a navigation bar with various links and a 'COMENZAR GRATIS' button. The main area is titled 'Selecciona un proyecto' (Select a project) and shows a list of existing projects: 'Asistente Virtual', 'My Project', and 'App Enerdrails'. Below this, there's a 'PROYECTO NUEVO' (New project) button. Further down, the 'Proyecto nuevo' (New project) creation screen is displayed, asking for the project name ('Nombre del proyecto') which is set to '1-PlatziBlog DB'. There are also fields for 'Ubicación' (Location) and buttons for 'CREAR' (Create) and 'CANCELAR' (Cancel).

Ya que se haya creado y seleccionado el proyecto de Servicios Administrados del Platzi Blog, daremos clic en el menú de la esquina superior izquierda y seleccionaremos la opción de SQL.

The screenshot shows the Google Cloud Console dashboard for the '1-PlatziBlog DB' project. The left sidebar menu is open, showing various Google Cloud services: Facturación, IAM y administración, Marketplace, Compute Engine, Kubernetes Engine, Cloud Storage, BigQuery, Red de VPC, Cloud Run, and SQL. The 'SQL' option is highlighted. The main content area has a welcome message 'Te damos la bienvenida' (Welcome) and several quick access buttons: 'Crea una VM', 'Ejecuta una consulta en BigQuery', 'Crea un clúster de GKE', and 'Crea un bucket de almacenamiento'.

Como el servicio de base de datos de Google Cloud, después de cierto número de datos empieza a cobrar, pedirá que ingreses tus datos bancarios, aunque en un inicio no cobrará nada, solo hasta pasar cierto umbral de número de datos almacenados. Ya que se hayan ingresado, daremos clic al botón de Crear Instancia con Créditos Gratuitos → MySQL → Habilitar API.



### Elige tu motor de base de datos

MySQL Versiones: 8.0, 5.7, 5.6 <a href="#">Elegir MySQL</a>	PostgreSQL Versiones: 15, 14, 13, 12, 11, 10, 9.6 <a href="#">Elegir PostgreSQL</a>	SQL Server Versiones: 2022, 2019, 2017 <a href="#">Elegir SQL Server</a>
---	---	--

[¿Quieres obtener más información sobre los motores de base de datos de Cloud SQL?](#)  
[Más información](#)

Ahora daremos clic en Instance ID para asignar un nombre a la base de datos y luego indicaremos su contraseña, los demás parámetros como zona y región se refieren a la ubicación del servidor, que por el momento no es de suma importancia. Además, me obliga a que elija un plan de cobro dependiendo del servicio, pero cabe mencionar que la versión de prueba incluye 300 USD en forma de créditos, por lo que se deberá monitorear el uso de la DB, porque en máximo 90 días, podría empezar a realizar cobros.

Resumen	
Edición de Cloud SQL	Enterprise
Region	us-central1 (Iowa)
Versión de la base de datos	MySQL 8.0
CPU virtuales	2 CPU virtual(es)
Memoria	8 GB
Caché de datos	Inhabilitada
Almacenamiento	10 GB
Conexiones	IP pública
Copia de seguridad	Automatizada
Disponibilidad	Zona única
Recuperación de un momento determinado	Habilitada

Ya que haya dado clic al botón de CREAR INSTANCIA, aparecerá el dashboard de la base de datos MySQL, pero en este momento no tendrá ninguna base de datos creada, por lo cual deberemos acceder a ella en la parte lateral izquierda donde dice Bases de datos → CREAR BASE DE DATOS → Asignar nombre y codificación (utf8) → CREAR. Para que se pueda completar este proceso, deberemos dejarla acabar de crear la instancia.

The screenshot shows the Google Cloud SQL dashboard for the '1-Platziblog DB' project. On the left sidebar, under 'INSTANCIA PRINCIPAL', the 'Bases de datos' option is selected. In the main content area, a new instance 'platziblog-db' is being created. A message at the top states: 'Se está creando la instancia. Este proceso puede demorar algunos minutos. Mientras tanto, puedes seguir viendo información sobre la instancia.' Below this, there's a chart titled 'Uso de CPU' which displays a timeline from UTC-6 18:00 to 16:00. A note says: 'No hay datos disponibles para el período seleccionado.' At the bottom of the instance card, there's a link: 'Ir a Estadísticas de consultas para obtener información más detallada sobre las consultas y el rendimiento'.

The screenshot shows the 'Crear una base de datos' dialog box. It has fields for 'Nombre de la base de datos' (set to 'platziblog-db'), 'Intercalación' (set to 'utf8'), and 'Grupo de caracteres' (set to 'utf8'). There are also dropdowns for 'Intercaleación' (set to 'Intercaleación predeterminada') and 'Más información'. At the bottom are 'CREAR' and 'CANCELAR' buttons.

Ahora volveremos a la pestaña principal de Descripción general para bajar en la ventana y dar clic en el botón de Conectarse a esta instancia → ABRIR CLOUD SHELL.

The screenshot shows the 'Descripción general' tab of the Google Cloud SQL dashboard. Under 'INSTANCIA PRINCIPAL', the 'Conectar a esta instancia' section is highlighted. It contains fields for 'Dirección IP pública' (34.171.9.248) and 'Nombre de la conexión' (platziblog-db:484922:us-central1:platziblog-db). Below this, a note says: '¿Necesitas ayuda para conectarte?' and provides links to documentation. To the right, a 'Configuración' panel shows resource allocation: 'CPU virtuales' (2), 'Memoria' (8 GB), and 'Almacenamiento de SSD' (10 GB). At the bottom right, a 'Operaciones de1-Platziblog DB' panel shows a log entry: 'Se reinició platziblog-db' at '17:01:15 GMT-6'.

Con esta terminal nos podremos comunicar con la base de datos recién creada, donde por default se nos proporciona el comando con el cual nos podemos conectar con la base de datos, que es el siguiente:

```
gcloud sql connect platziblog-db --user=root --quiet
```

Pero antes de poder acceder a ella, se debe activar una API que permite hacerlo y se encuentra en el siguiente link:

<https://console.cloud.google.com/apis/api/sqladmin.googleapis.com>

The screenshot shows the Google Cloud Platform interface for the Cloud SQL Admin API. At the top, there's a banner about a free trial credit. Below it, the navigation bar includes 'Google Cloud' and '1-PlatziBlog DB'. The main content area is titled 'Cloud SQL Admin API' and describes it as a 'Google Enterprise API' for managing Cloud SQL database instances. It features a 'PROBAR ESTA API' button. On the left, a sidebar shows 'INSTANCIA PRINCIPAL' with 'Descripción general' selected, along with other options like 'Estadísticas del sistema', 'Estadísticas de consultas', 'Conexiones', 'Usuarios', and 'Notas de versión'. The main panel has tabs for 'Descripción general', 'EDITAR', 'IMPORTAR', 'EXPORTAR', 'REINICIAR', 'DETENER', 'BORRAR', and 'CLONAR'. It displays system status from UTC-6 to UTC+0, and a 'Configuración' section with CPU virtuales (2), Memoria (8 GB), and Almacenamiento de SSD (10 GB). Below this is a 'Conectar a esta instancia' section with an IP address input field containing '34.171.9.248'. A 'Terminal' tab in the bottom right shows a Cloud Shell session with the command: 'Welcome to Cloud Shell! Type "help" to get started. Your Cloud Platform project in this session is set to platziblog-db-404922. Use "gcloud config set project [PROJECT\_ID]" to change to a different project. diegorko06@cloudshell:~\$ gcloud sql connect platziblog-db --user=root --quiet'.

Cabe mencionar que la conexión solo durará 5 minutos por cuestiones de seguridad, al acabar el proceso de conexión se introducirá la contraseña y ya tendremos acceso a la base de datos.

This screenshot shows the same Google Cloud Platform interface as the previous one, but the Cloud Shell terminal output now shows a successful MySQL connection. The command 'gcloud sql connect platziblog-db --user=root --quiet' was run, followed by the MySQL prompt 'mysql>'. The terminal also displays copyright information for Oracle and a note about Oracle trademarks.

# Lenguaje de Programación SQL

Las siglas SQL de significan Structured Query Language, la función principal de este lenguaje de programación es la de realizar consultas a datos de una forma estandarizada no importando que base de datos se esté utilizando.

Además del lenguaje SQL existen los lenguajes NoSQL, cuyas siglas significan Not Only SQL, esto se utiliza más que nada en bases de datos no relacionales, donde, aunque sí se sigue utilizando en algunas partes el lenguaje SQL, se utilizan algunas variantes de este lenguaje. Algunos ejemplos de las bases de datos no relacionales que utilizan alguna variante de SQL son Cassandra, Big Query, etc.

## Sub-lenguajes de SQL: DDL (Data Definition Language)

Los dos sub-lenguajes más importantes del lenguaje SQL son llamados DDL y DML, primero se explicará el DDL que sirve para estructurar la base de datos:

- **DDL (Data Definition Language):** La función principal de este sub-lenguaje de SQL es la de crear la estructura de una base de datos. Esto se refiere a las entidades, atributos, relaciones, etc. que son descritos en un diagrama ER o en un diagrama físico. Los **3 comandos** con los que cuenta para llevar a cabo la estructuración de datos con el lenguaje DDL son los siguientes:
  - **CREATE:** Comando que ayuda a crear una base de datos, tabla, vista, índice, etc.
  - **ALTER:** Comando que ayuda a alterar o modificar una entidad, tabla, tipo de dato, etc.
  - **DROP:** Comando que ayuda a eliminar elementos de una base de datos. Esta se debe utilizar con mucho cuidado, ya que accidentalmente podríamos borrar nuestra db completa.

**Algunos de los elementos u objetos** que se van a manipular con los **comandos DDL** son los siguientes:

- **DATABASE:** Se refiere a la base de datos, a veces también se le llama **SCHEMA**.
- **TABLE:** Las tablas se refieren a la visualización de los datos después de haberlas modelado a través de un diagrama ER o diagrama físico.
- **VIEW:** Las vistas se refieren a la forma en la que se pueden interpretar y ordenar los datos extraídos de una database, de tal forma que puedan ser entendidos o de utilidad para el usuario, ya que, en las tablas de las bases de datos, muchas veces la información está segmentada y de esta forma la podemos filtrar y organizar.

### CREATE:

#### *Crear una Base de Datos con SQL*

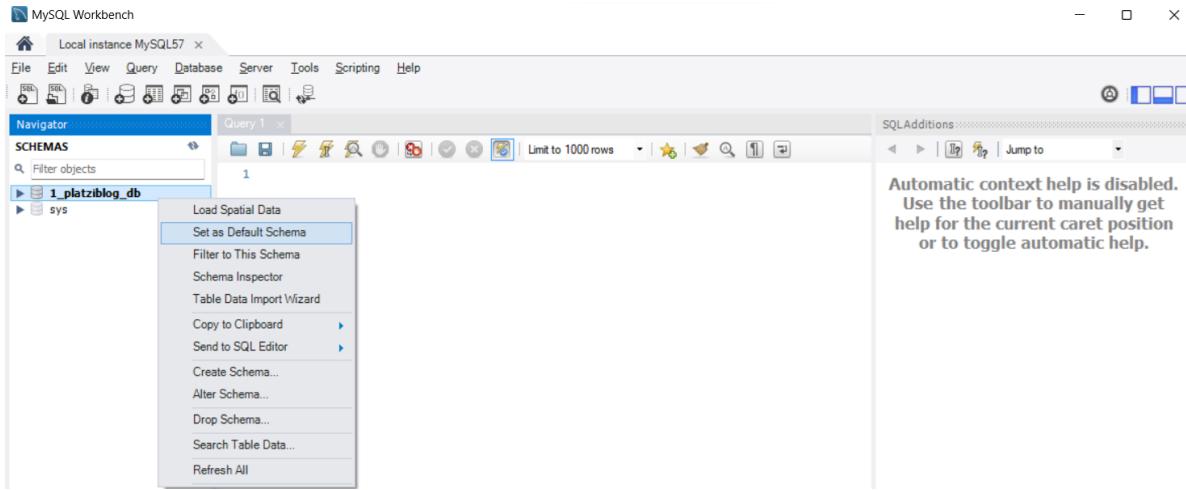
Siempre que se quiera crear una base de datos nueva se deberán ejecutar los siguientes comandos, ya sea en un Manejador de Bases de Datos Relacionales (RDBMS) o en una Base de Datos con Servicios Administrados (Nube o Cloud):

**CREATE DATABASE**      "Nombre\_Base\_de\_Datos";

--La siguiente instrucción no siempre se incluye, porque se da por entendida.

```
USE      DATABASE      "Nombre_Base_de_Datos";
```

Para asegurarnos que el código SQL se está aplicando a una base de datos específica en MySQL Workbench, daremos clic derecho sobre ella y seleccionaremos la opción de Set as Default Schema.

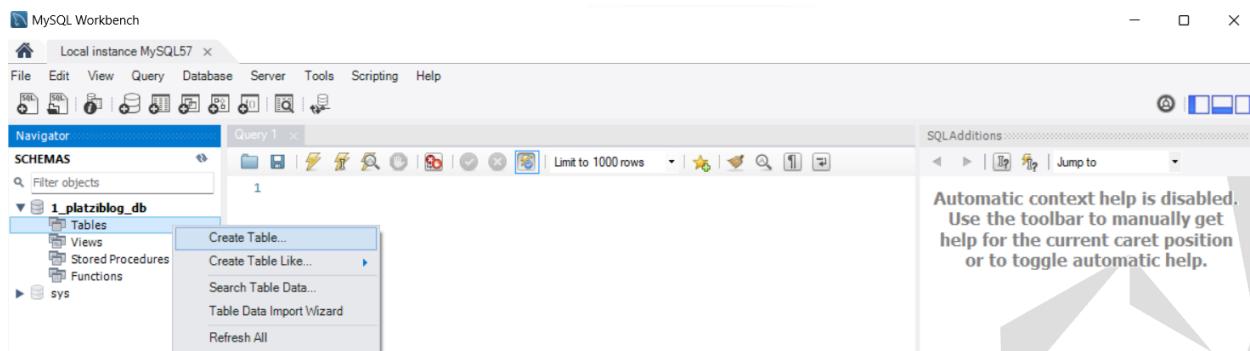


### **Crear una Tabla con SQL**

Cuando se quiera crear una nueva tabla en una base de datos se deberá ejecutar el siguiente comando, ya sea en un Manejador de Bases de Datos Relacionales (RDBMS) o en una Base de Datos con Servicios Administrados (Nube o Cloud), aunque cabe la pena mencionar que es importante previamente tener ya listo el diagrama ER y físico de la base de datos, para seguir dicha estructura:

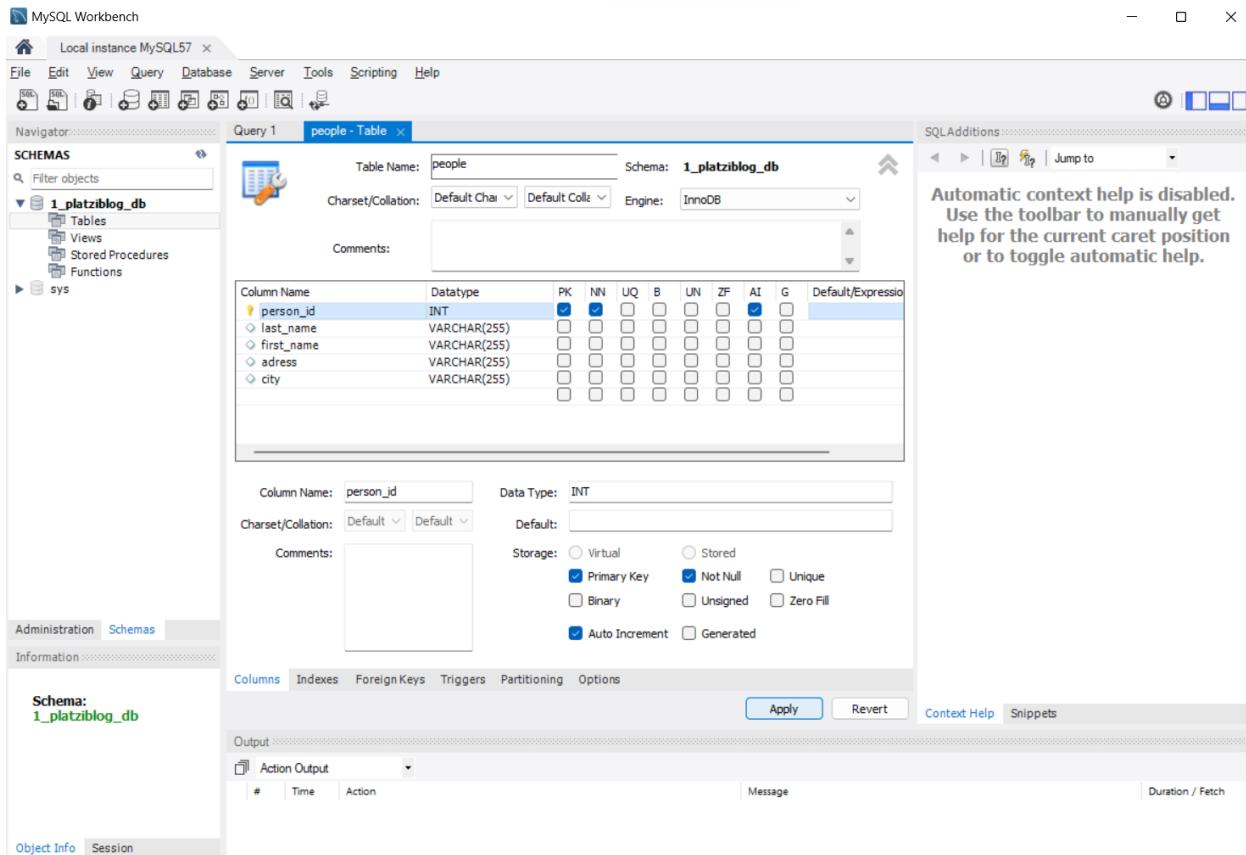
```
CREATE TABLE Nombre_Entidad_o_Tabla(  
    Nombre_Atributo     Tipo_de_Dato_y_Constraints ,  
    Nombre_Atributo     Tipo_de_Dato_y_Constraints  
);
```

Para crear una nueva tabla en una base de datos específica con código SQL en MySQL Workbench, desplegaremos la pestaña de la base de datos, daremos clic derecho sobre donde dice Tables y seleccionaremos la opción de Create Table...

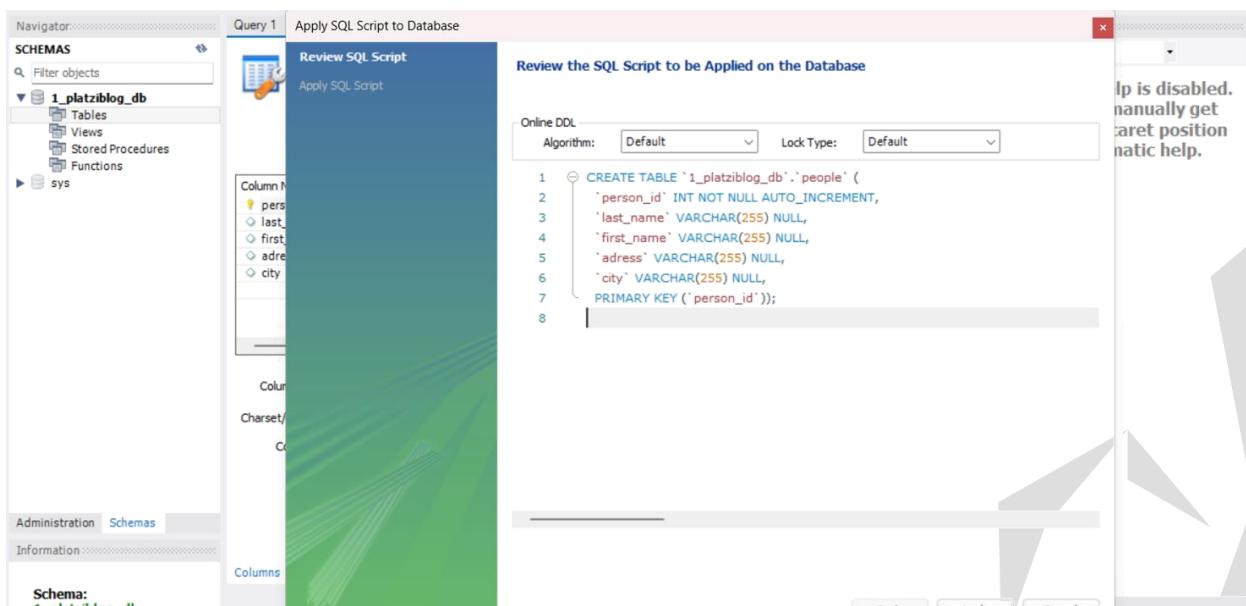


La ventaja de utilizar un cliente gráfico es que se puede visualizar el resultado del código.

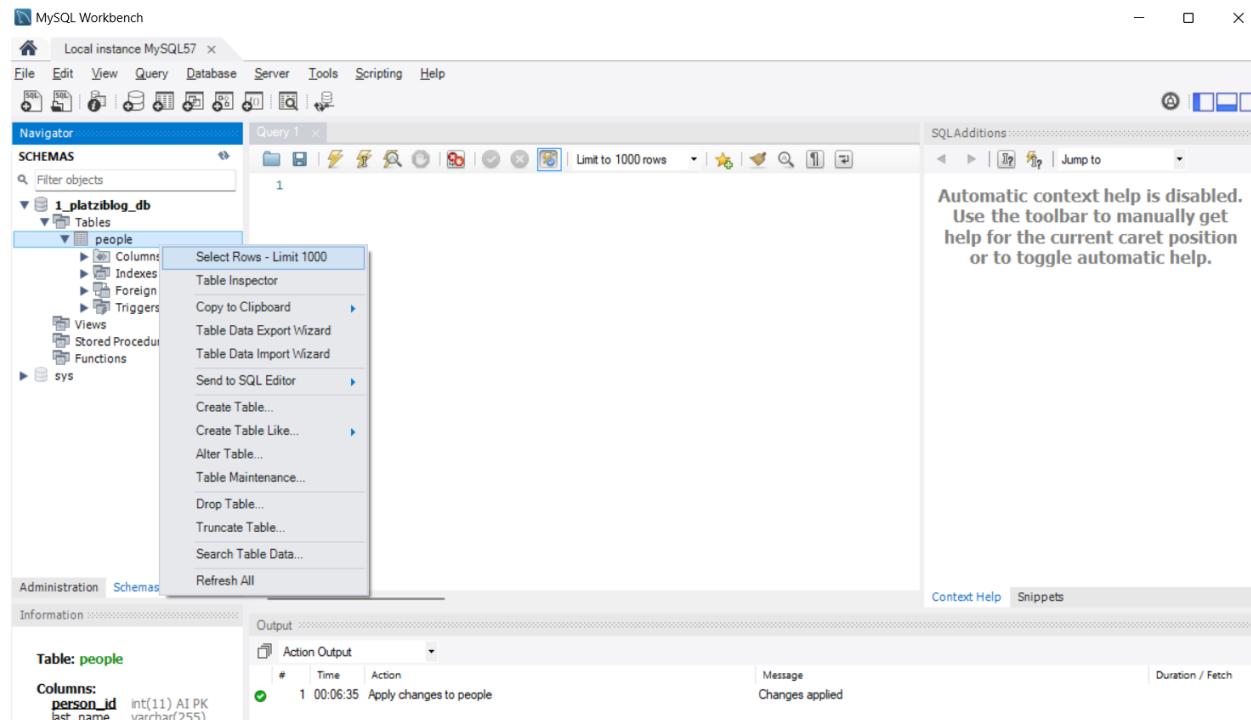
Ya que se hayan llenado todos los atributos de la tabla (entidad) que queremos crear, incluyendo sus **constraints (restricciones)**, daremos clic en el botón de Apply.



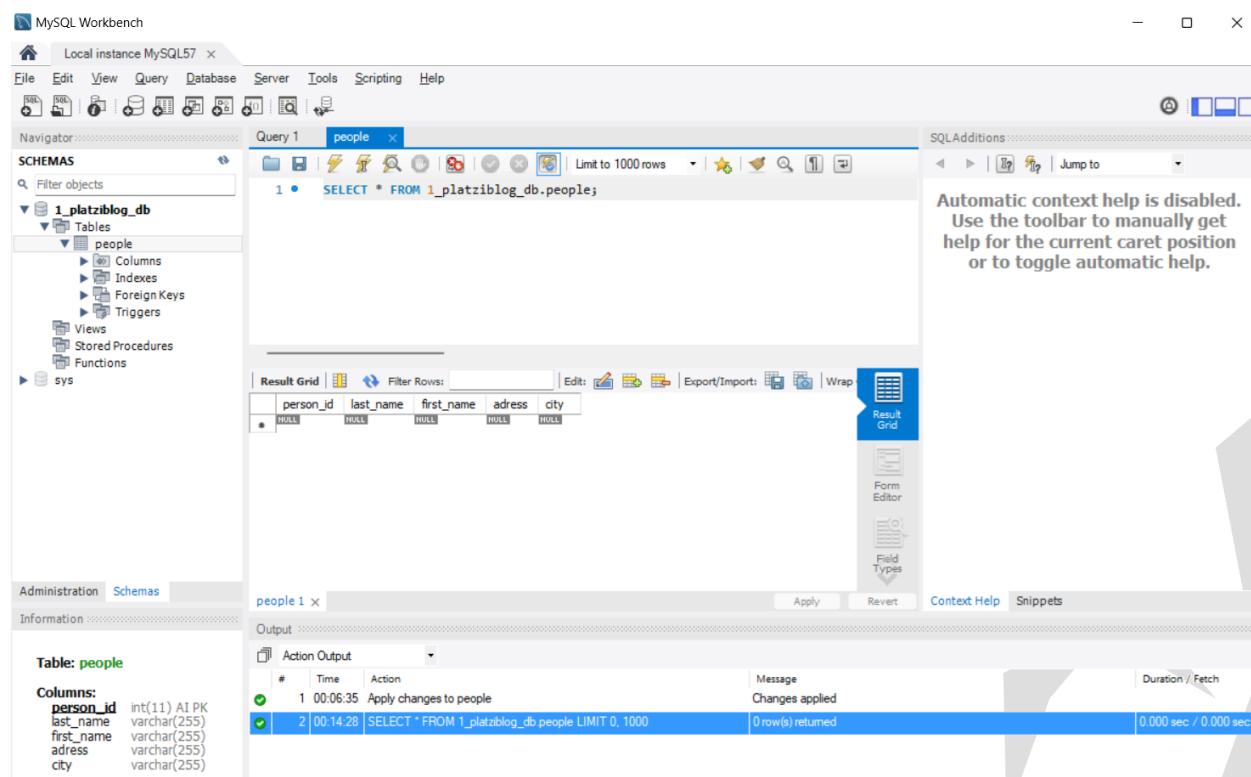
En este punto es donde veremos el código SQL que se ejecutará en la base de datos justo antes de que lo haga, y podremos observar que cumple la sintaxis descrita anteriormente. Ya que estemos satisfechos con el código mostrado, daremos de nuevo clic en el botón de Apply → Finish.



Ahora ya podremos visualizar los datos almacenados en esta **entidad** al ingresar dentro del desplegable llamado Tables, dar clic derecho sobre el nombre de la **tabla** creada y seleccionar la opción de Select Rows - Limit 1000.



Y veremos la **tabla** con los **atributos** que le agregamos en un formato de tabla.



### **Crear una Vista con SQL**

Cuando se quiera crear una nueva vista en una base de datos, la cual representa una extracción de datos que se encuentren filtrados y organizados, se deberá ejecutar el siguiente comando, ya sea en un Manejador de Bases de Datos Relacionales (RDBMS) o en una Base de Datos con Servicios Administrados (Nube o Cloud):

```
USE      DATABASE      "Nombre_Base_de_Datos";  
CREATE OR REPLACE  VIEW    "v_Nombre_Vista" AS  
//Filtro o tabla de la que se quieran obtener los datos de la vista.
```

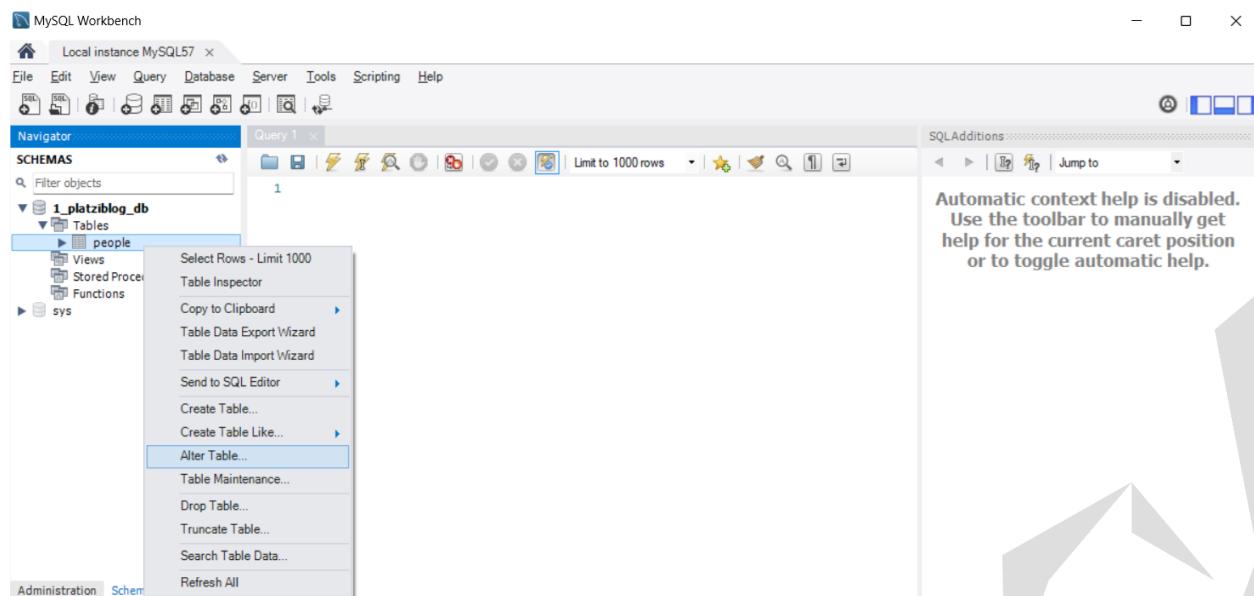
ALTER:

### **Alterar una Tabla con SQL**

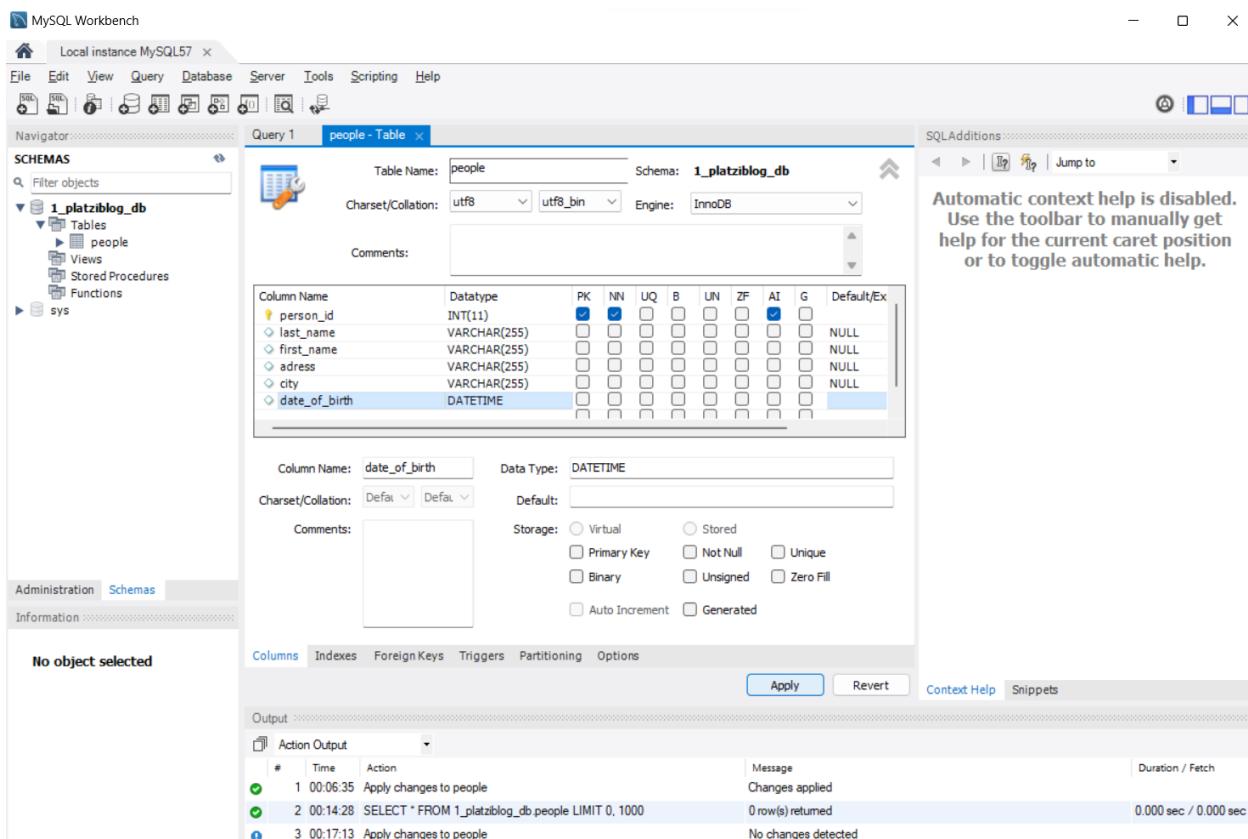
Cuando se quiera modificar una tabla en una base de datos se deberá ejecutar el siguiente comando, ya sea en un Manejador de Bases de Datos Relacionales (RDBMS) o en una Base de Datos con Servicios Administrados (Nube o Cloud):

```
ALTER  TABLE      "Nombre_Base_de_Datos". "Nombre_Entidad_o_Tabla";  
ADD    COLUMN     "Nombre_Atributo"      Tipo_de_Dato_y_Constraints;  
CHANGE COLUMN    "Columna_a_Modificar"  Tipo_de_Dato_y_Constraints;  
DROP    COLUMN     "Columna_a_Borrar";
```

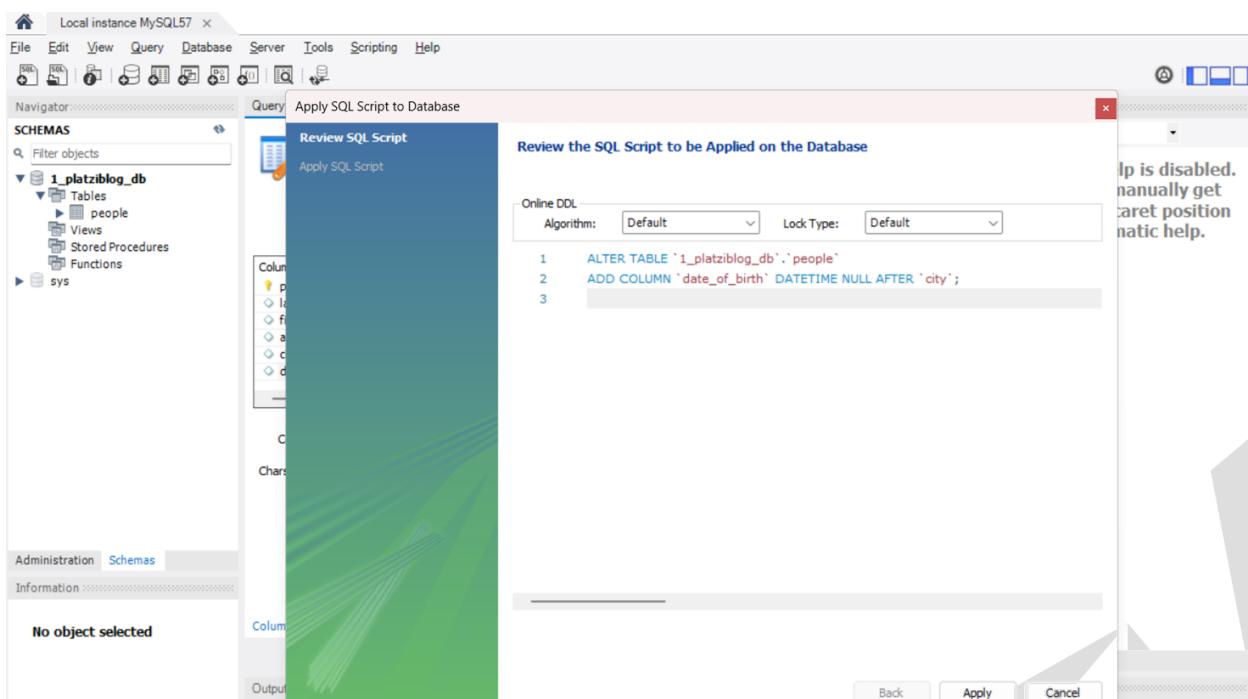
Para alterar una **entidad o tabla** en una base de datos específica con código SQL en MySQL Workbench, desplegaremos la pestaña de la base de datos → luego la de Tables → Daremos clic derecho sobre la entidad que se quiere modificar → Seleccionaremos la opción de Alter Table...



Todas las acciones de agregar, editar y borrar se ejecutarán de forma manual dentro del RDBMS.



Y al dar clic en el botón de Apply podremos ver su equivalente en formato de código SQL.



Recordemos que para ver el resultado en formato de tabla se debe dar clic derecho sobre el nombre de la **tabla** y seleccionar la opción de Select Rows - Limit 1000.

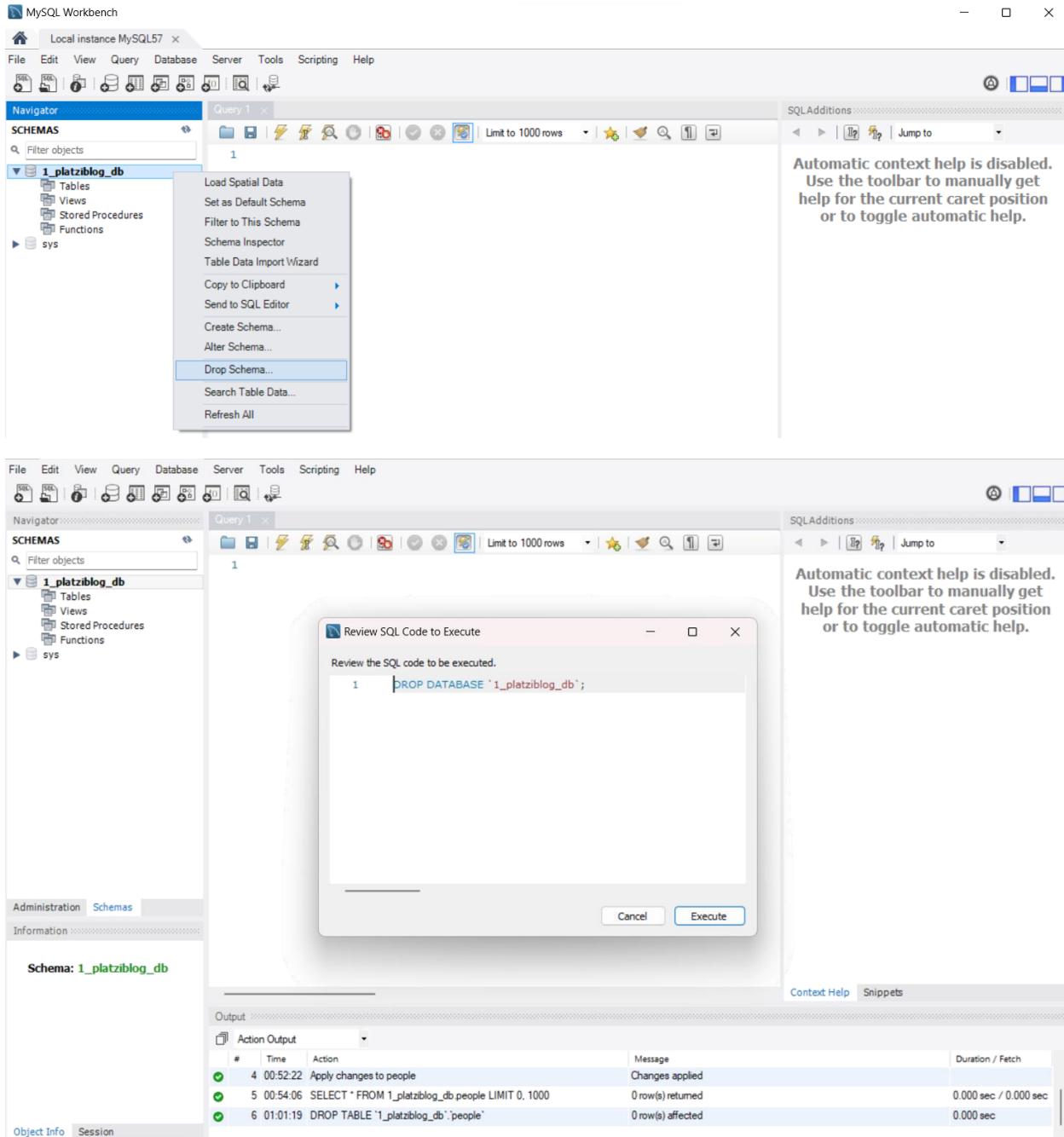
**DROP:**

### Borrar una Tabla, Columna o Base de Datos con SQL

Cuando se quiera modificar una tabla en una base de datos se deberá ejecutar el siguiente comando, ya sea en un Manejador de Bases de Datos Relacionales (RDBMS) o en una Base de Datos con Servicios Administrados (Nube o Cloud):

```
DROP DATABASE Nombre_Base_de_Datos;
DROP TABLE Nombre_Entidad_o_Tabla;
DROP COLUMN Columna_a_Borrar;
```

La misma secuencia que se aplica para alterar una tabla se aplica para borrar columnas, tablas o db.



## Sub-lenguajes de SQL: DML (Data Manipulation Language)

Los dos sub-lenguajes más importantes del lenguaje SQL son llamados DDL y DML, ya se abordó el lenguaje DDL que sirve para estructurar la base de datos y ahora se explicará el DML que sirve para manipular los datos dentro de ella:

- **DML (Data Manipulation Language):** La función principal de este sub-lenguaje de SQL es la de manipular los datos de una base de datos, ya sea para crear vistas o para meter, actualizar, borrar o extraer datos. Los **4 comandos** con los que cuenta para llevar a cabo la manipulación de datos con el lenguaje DML son los siguientes:

- **INSERT**: Comando que permite introducir nuevas filas de datos (también llamados registros o tuplas) a la tabla (entidad) de una base de datos.
- **UPDATE**: Comando que actualiza o modifica datos ya existentes en la tabla de una base de datos.
- **DELETE**: Instrucción que sirve para borrar toda la fila de datos de una tabla perteneciente a una base de datos.
- **SELECT**: Este comando es muy importante, ya que permite traer información de la base de datos, ya sea para crear una vista, extraerla para ponerla en otro lado, etc.

Con los comandos descritos se pueden realizar todas las acciones del CRUD (acrónimo de Create, Read, Update, Delete), que es un conjunto de operaciones básicas realizadas en cualquier sistema de gestión de bases de datos.

## INSERT:

### *Insertar Datos Nuevos a la Tabla de una Base de Datos con SQL*

Cuando se quiera añadir nuevos datos a la fila de la tabla de una base de datos se ejecutará el comando **INSERT** de la siguiente manera:

```
INSERT INTO Nombre_Entidad_o_Tabla(
    Nombre_Atributo_o_Columna_1,
    Nombre_Atributo_o_Columna_2,
    Nombre_Atributo_o_Columna_n
)
VALUES(
    "Valor_Atributo_o_Columna_1",
    "Valor_Atributo_o_Columna_2",
    "Valor_Atributo_o_Columna_n"
);
```

Cuando se ejecute el código SQL al dar clic en el rayo que se encuentra en la parte superior, se añadirán los datos indicados con el método **VALUES** en las columnas indicadas dentro del paréntesis de la instrucción **INSERT**.



Podemos visualizar los datos añadidos al ingresar en el desplegable Tables → Dar clic derecho sobre el nombre de la **tabla** creada → Seleccionar la opción de Select Rows - Limit 1000.

## UPDATE:

### *Editar Datos en la Tabla de una Base de Datos con SQL*

Cuando se busque editar los datos de la tabla de una base de datos se deberá indicar exactamente a que posición o posiciones de la tabla nos estamos refiriendo, ya que el cambio se puede realizar en una o varias filas de la entidad. Para ello se utiliza el comando **SET** seguido del nombre del **atributo** que se quiere modificar y el nuevo **valor** que adoptará, luego se señala la fila por medio de la instrucción **WHERE** acompañada de algún valor que la identifique, para así indicar si se busca modificar solo una fila en específico o todas las filas donde se cumpla esta condición:

```
UPDATE "Nombre_Entidad_o_Tabla"
SET      "Columna_1" = "Valor_Columna_1", "Atributo_2" = "Valor_Atributo_2"
WHERE   "Nombre_Atributo_o_Columna" = "Valor_Fila_o_Identificador";
```

Cabe mencionar que, si no se indica nada en la instrucción **WHERE**, todos los datos de la columna indicada serán editados al nuevo valor indicado. Pero al intentar esto es cuando se vuelve evidente el uso de un manejador de base de datos, ya que cuando se intente realizar una edición masiva de datos al dar clic en el rayo que se encuentra en la parte superior, esto será evitado por el programa, evitando así que se realicen cambios indeseables.

Podemos visualizar los datos modificados al ingresar en el desplegable Tables → Dar clic derecho sobre el nombre de la **tabla** creada → Seleccionar la opción de Select Rows - Limit 1000 o simplemente dando de nuevo clic en el botón de rayo.

person_id	last_name	first_name	address	city
1	Chávez	Laura	Calle 21	Mérida

## DELETE:

### **Borrar Todos los Datos de una Fila Perteneciente a la Tabla de una Base de Datos con SQL**

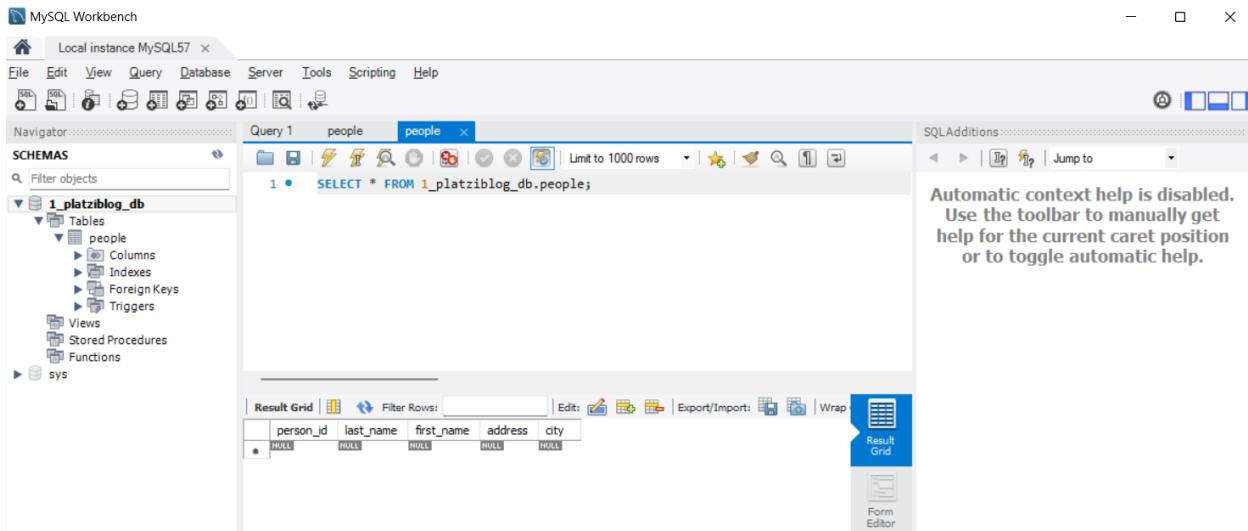
Cuando se busque borrar datos de una tabla, esto se hará en toda la fila de ese registro o tupla. Para ello se utiliza el comando **WHERE** acompañado de algún valor que identifique la fila o filas que se busca eliminar:

```
DELETE FROM "Nombre_Entidad_o_Tabla"
WHERE "Nombre_Atributo_o_Columna" = "Valor_Fila_o_Identificador";
```

Cabe mencionar que, si no se indica nada en la instrucción **WHERE**, todos los datos de la tabla indicada serán eliminados. Por lo que hay que tener mucho cuidado cuando se utilice este comando, ya que podríamos borrar accidentalmente todos los datos de nuestra base de datos.

```
1 • DELETE FROM people
2 WHERE person_id = 1
```

Podemos visualizar los datos eliminados al ingresar en el desplegable Tables → Dar clic derecho sobre el nombre de la **tabla** creada → Seleccionar la opción de Select Rows - Limit 1000 o simplemente dando de nuevo clic en el botón de rayo.



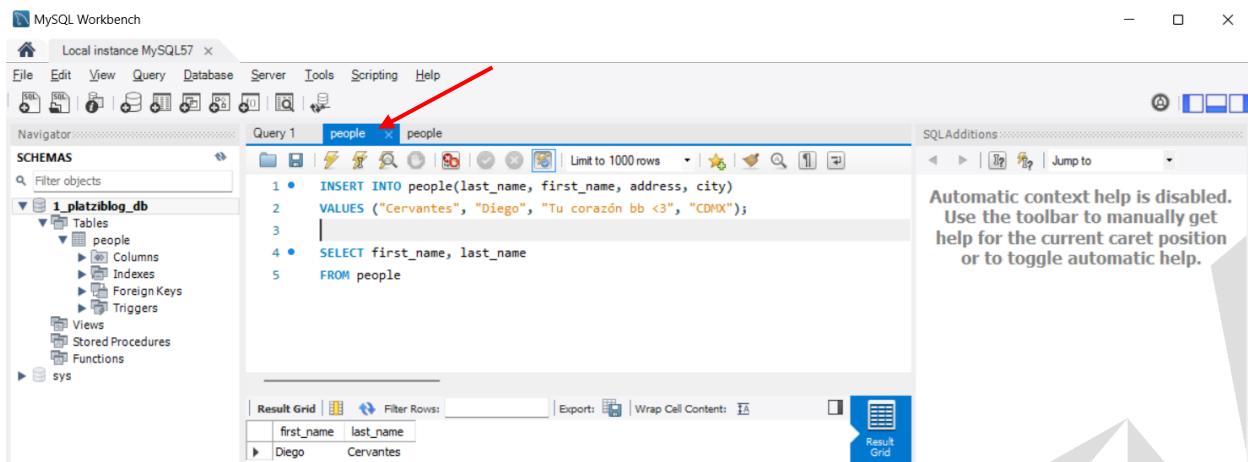
**SELECT:**

### **Extraer Todos los Datos de una Columna Perteneciente a la Tabla de una Base de Datos**

Cuando se busque obtener todos los datos pertenecientes a la **columna (atributo)** de una **tabla**, ya sea para crear una **vista** o para simplemente moverlos a otro lado, se utiliza el comando **SELECT**, donde se debe indicar el nombre del **atributo** o **columna** de datos que se quiere extraer y a qué **tabla** o **entidad** pertenezcan:

```
SELECT Nombre_Atributo_o_Columna_1, Nombre_Atributo_o_Columna_n
FROM Nombre_Entidad_o_Tabla;
```

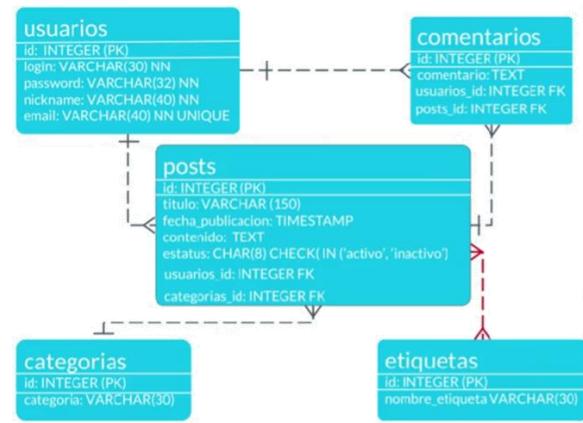
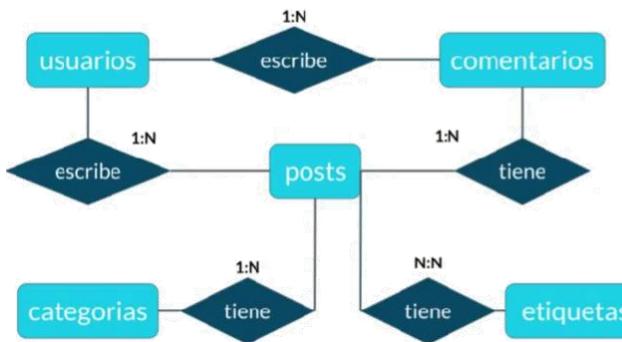
Cabe mencionar que, si no se indica una columna o atributo en específico y en su lugar se utiliza un asterisco, se extraerán todas las columnas de la tabla indicada. Para ejecutar el código SQL en la base de datos se debe dar clic en el botón de rayo dentro de MySQL Workbench.



# Ejemplo de Diagrama Físico a Código SQL

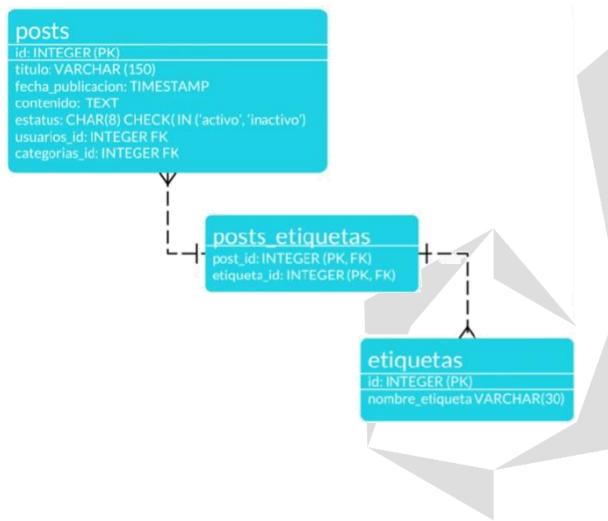
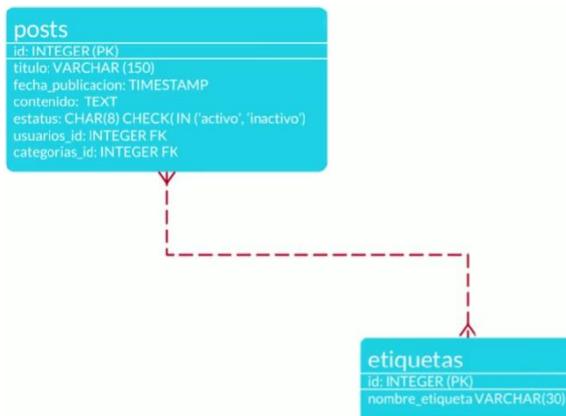
Se tenía previamente hecho el **diagrama ER (entidad-relación)** y el **diagrama físico** de una base de datos relacional cuya intención era almacenar los datos obtenidos a través de un blog. En ambos diagramas se podían observar todas las **entidades (tablas)** y **relaciones (conexiones)** que conforman a la base de datos y en el diagrama físico hasta se describía el **tipo de dato** de cada uno de sus **atributos (columnas de la tabla)**.

## Diagrama ER: Platziblog



El propósito de esta parte es traducir estos diagramas a código SQL, a continuación, describiremos los pasos para lograr dicho cometido:

1. Primero se codifican las **entidades independientes**, osea las que no tienen ninguna **llave foránea**, con esto nos referimos a las **tablas** que tienen una **relación** con **cardinalidad de 1**, en el **diagrama ER** esto se observa de forma muy clara, pero en el diagrama físico esto se denota por el lado de la conexión donde se tiene una raya, no un triángulo.
  - a. Recordemos que la **cardinalidad** se refiere al **número de tablas de datos** que se pueden desprender de otra **tabla de datos**, por ejemplo, se puede decir que **un usuario puede tener varios comentarios**, pero **un comentario, no puede venir de varios usuarios**, por lo cual esa **relación** tendrá una **cardinalidad** de **1:N**.
    - i. Aquí vale la pena recordar que cuando se tiene una **relación con cardinalidad N:N** diferenciada por una tonalidad roja, se deberá añadir una tabla interna que relacione ambos id entre sí para crear un identificador único.



- b. La llave foránea de igual manera se refiere a la dirección de conexión denotada por las **restricciones PRIMARY KEY y FOREIGN KEY**, que las **relacionan**.

## Constraints (Restricciones)

Constraint	Descripción
NOT NULL	Se asegura que la columna no tenga valores nulos
UNIQUE	Se asegura que cada valor en la columna no se repita
PRIMARY KEY	Es una combinación de NOT NULL y UNIQUE
FOREIGN KEY	Identifica de manera única una tupla en otra tabla
CHECK	Se asegura que el valor en la columna cumpla una condición dada
DEFAULT	Coloca un valor por defecto cuando no hay un valor especificado
INDEX	Se crea por columna para permitir búsquedas más rápidas

Por esta situación es que en este ejemplo primero se codificarán las tablas de **usuarios** y **categorías**:

Online DDL

Algorithm: Default Lock Type: Default

```

1  CREATE TABLE `1_platziblog_db`.`categorias` (
2      `id` INT NOT NULL AUTO_INCREMENT,
3      `nombre_categoria` VARCHAR(30) NOT NULL,
4      PRIMARY KEY (`id`));
5

```

Online DDL

Algorithm: Default Lock Type: Default

```

1  CREATE TABLE `1_platziblog_db`.`usuarios` (
2      `id` INT NOT NULL AUTO_INCREMENT,
3      `login` VARCHAR(30) NOT NULL,
4      `password` VARCHAR(32) NOT NULL,
5      `nickname` VARCHAR(40) NOT NULL,
6      `email` VARCHAR(40) NOT NULL,
7      PRIMARY KEY (`id`),
8      UNIQUE INDEX `email_UNIQUE` (`email` ASC));

```

2. Ahora se codificarán las **entidades dependientes**, osea las que sí tienen conexión con alguna llave foránea o **relación** con **cardinalidad de N**. Para este paso debemos ver el **orden de las dependencias en las tablas**, partiendo primero de las **entidades cuyas dependencias ya se hayan codificado**.

Por esta situación es que en este paso primero se codificarán con SQL las tablas de **posts** (que **depende de usuarios y categorías, las cuales ya están codificadas**) y luego se realizará la tabla de **comentarios** (que **depende a su vez de posts y usuarios**):

Aquí vale la pena mencionar que, si dentro de la columna Default/Expression al crear la tabla se pone un valor entre comillas, cuando ese atributo de la tabla no reciba ningún valor, adoptará por default ese.

The screenshot shows the MySQL Workbench interface. In the central pane, a table named 'posts' is being created under schema '1\_platziblog\_db'. The 'estatus' column is defined with a data type of CHAR(8), a default value of 'activo', and a comment 'Activo'. A red arrow points to the 'Default/Expression' field in the column definition grid. Another red arrow points to the 'activo' value in the same row of the grid.

Ahora ya podremos crear nuestra tabla dando clic en el botón de Apply, pero después debemos asegurarnos de que las id que provienen de otras tablas sean del mismo tipo para asegurarnos de su conexión.

Online DDL

Algorithm: Default Lock Type: Default

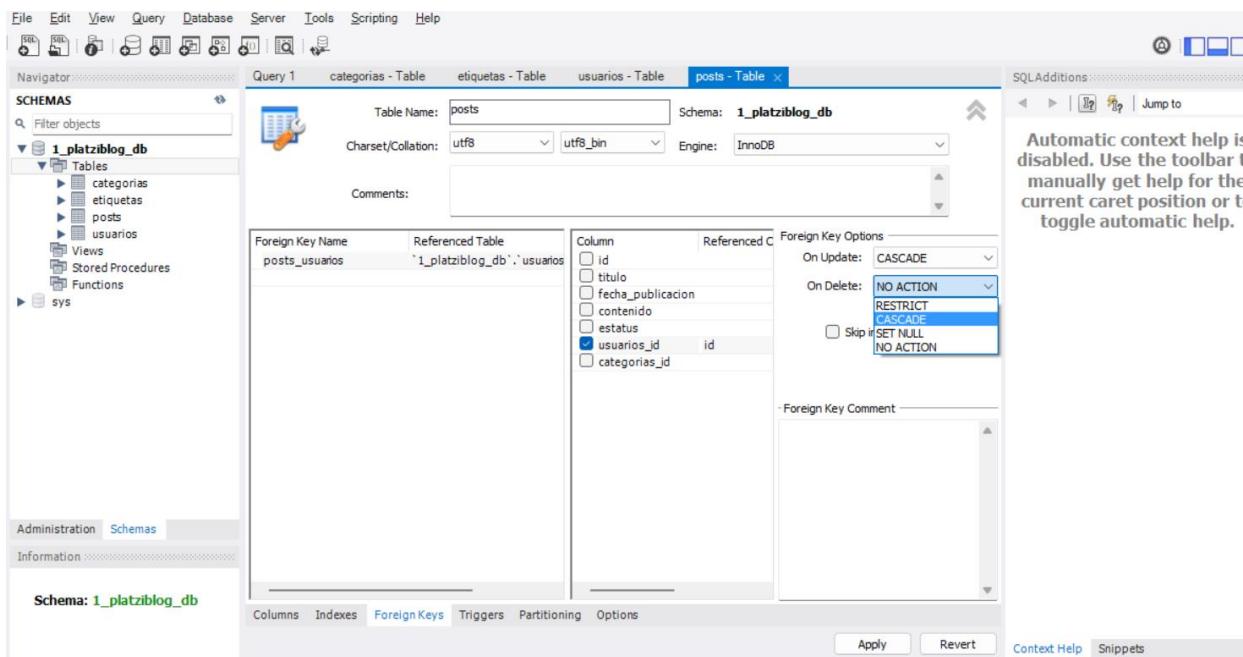
```

1  CREATE TABLE `1_platziblog_db`.`posts` (
2      `id` INT NOT NULL AUTO_INCREMENT,
3      `titulo` VARCHAR(150) NOT NULL,
4      `fecha_publicacion` TIMESTAMP NULL,
5      `contenido` TEXT NOT NULL,
6      `estatus` CHAR(8) NULL DEFAULT 'activo',
7      `usuarios_id` INT NOT NULL,
8      `categorias_id` INT NOT NULL,
9      PRIMARY KEY (`id`));
10

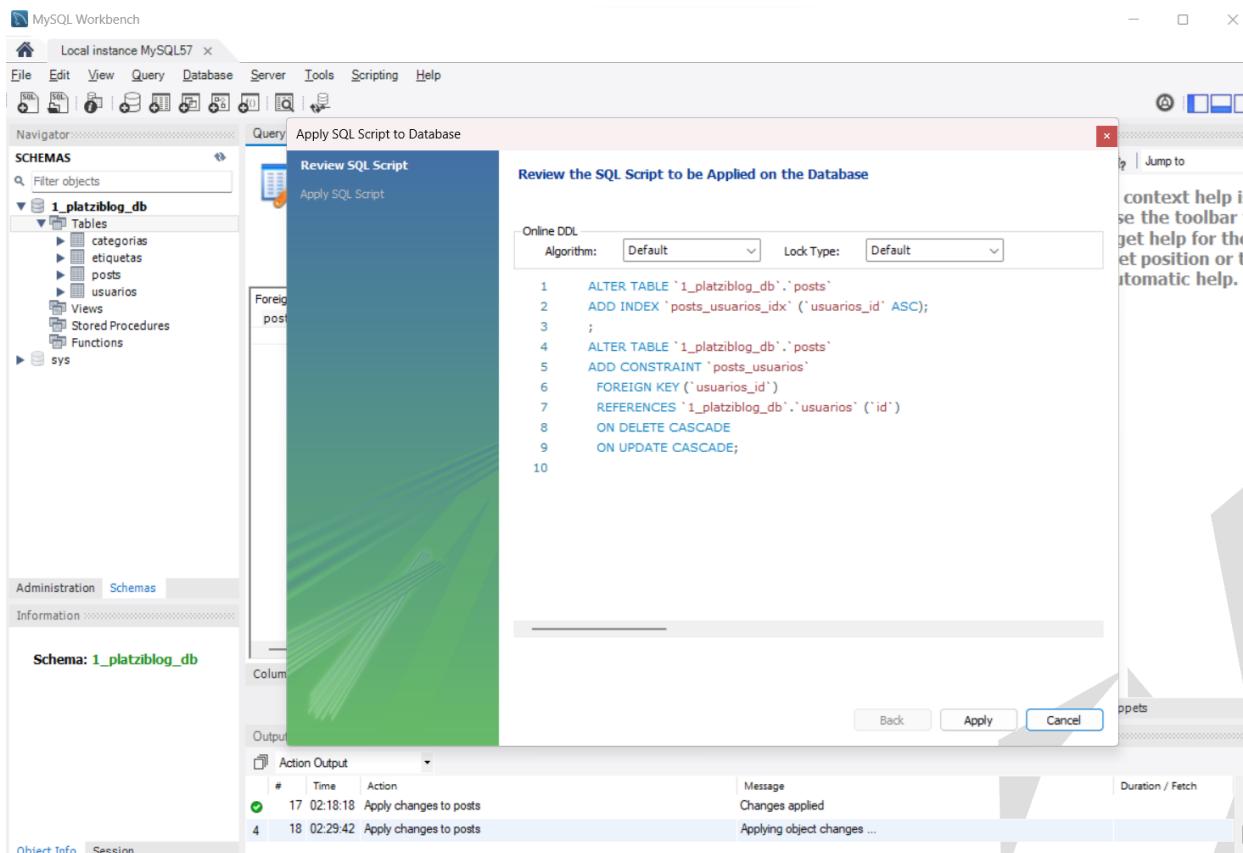
```

Para indicar las **llaves foráneas o índices** que **conectan dos tablas** nos introduciremos en la pestaña inferior que dice **Foreign Keys** → Después deberemos asignar un nombre a dicha foreign key (donde para indicar la conexión que se está realizando puede utilizar un nombre que combine los nombres de las dos tablas que combina) → Luego se deberá seleccionar la **entidad** de donde proviene la **relación** → Posteriormente deberemos elegir cuál **atributo** es el que realiza la **conexión** entre ambas **entidades** → A continuación se deberá indicar la acción a realizar en dicha conexión cuando se actualice o borre alguna

fila de datos de la tabla principal, en ambos casos es conveniente seleccionar la opción de **CASCADE**, para que el cambio se vea reflejado en ambas **entidades** → Finalmente se da clic en Apply para crear la **relación** a través de este **índice**.

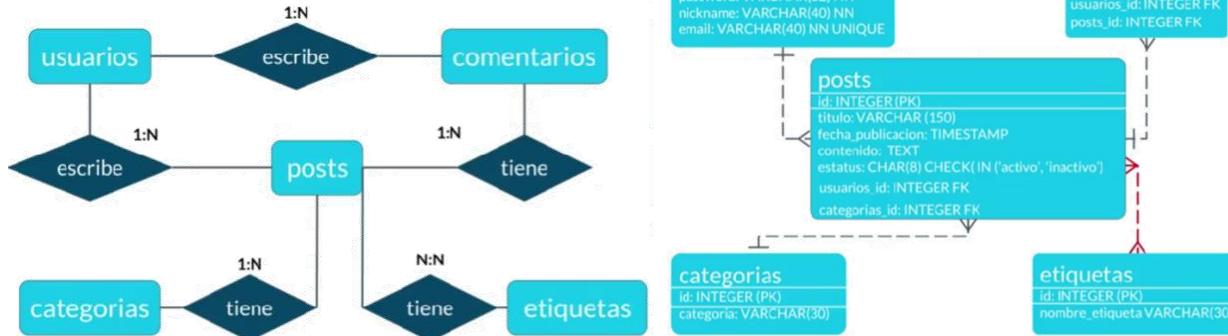


Cuando se cree un **índice**, de igual manera se creará un código SQL que denota dicha **relación**.



Ahora este mismo proceso se repite para crear el **índice de relación** entre las **tablas posts y categorías**:

## Diagrama ER: Platziblog



MySQL Workbench - Local instance MySQL57

File Edit View Query Database Server Tools Scripting Help

Navigator: Schemas: 1\_platziblog\_db Tables: categorias, etiquetas, posts, usuarios Views: Stored Procedures Functions sys

Table Name: posts Schema: 1\_platziblog\_db Charset/Collation: utf8 utf8\_bin Engine: InnoDB

Comments:

Foreign Key Name	Referenced Table	Column	Referenced C	Foreign Key Options
posts_usuarios	'1_platziblog_db'.usuarios	<input type="checkbox"/> id <input type="checkbox"/> titulo <input type="checkbox"/> fecha_publicacion <input type="checkbox"/> contenido <input type="checkbox"/> estatus <input type="checkbox"/> usuarios_id		On Update: CASCADE On Delete: CASCADE
posts_categorias	'1_platziblog_db'.categorias	<input checked="" type="checkbox"/> categorias_id	<input type="checkbox"/> id	Skip in SQL generation

Foreign Key Comment:

Columns: Indexes: Foreign Keys: Triggers: Partitioning: Options: Apply: Revert: Context Help: Snippets: Schema: 1\_platziblog\_db

### Online DDL

Algorithm: Default

Lock Type: Default

```

1   ALTER TABLE `1_platziblog_db`.`posts`
2     ADD INDEX `posts_categorias_idx` (`categorias_id` ASC);
3 ;
4   ALTER TABLE `1_platziblog_db`.`posts`
5     ADD CONSTRAINT `posts_categorias`
6       FOREIGN KEY (`categorias_id`)
7         REFERENCES `1_platziblog_db`.`categorias` (`id`)
8         ON DELETE CASCADE
9         ON UPDATE CASCADE;

```

Como se ha creado la **entidad** de **posts**, ya se puede crear la **tabla** de **comentarios** que depende de ella.

Online DDL

Algorithm: Default Lock Type: Default

```
1   CREATE TABLE `1_platziblog_db`.`comentarios` (
2     `id` INT NOT NULL AUTO_INCREMENT,
3     `cuerpo_comentario` TEXT NOT NULL,
4     `usuarios_id` INT NOT NULL,
5     `posts_id` INT NOT NULL,
6     PRIMARY KEY (`id`));
7
```

Una vez creada la **tabla**, se deberán crear sus **índices de relación** para denotar sus **llaves foráneas**, para ello se repite el proceso realizado en las conexiones de la **entidad posts**.

Online DDL

Algorithm: Default Lock Type: Default

```
1   ALTER TABLE `1_platziblog_db`.`comentarios`
2   ADD INDEX `comentarios_usuario_idx` (`usuarios_id` ASC);
3   ;
4   ALTER TABLE `1_platziblog_db`.`comentarios`
5   ADD CONSTRAINT `comentarios_usuario`
6     FOREIGN KEY (`usuarios_id`)
7     REFERENCES `1_platziblog_db`.`usuarios` (`id`)
8     ON DELETE CASCADE
9     ON UPDATE CASCADE;
10
```

Online DDL

Algorithm: Default Lock Type: Default

```
1   ALTER TABLE `1_platziblog_db`.`comentarios`
2   ADD INDEX `comentarios_posts_idx` (`posts_id` ASC);
3   ;
4   ALTER TABLE `1_platziblog_db`.`comentarios`
5   ADD CONSTRAINT `comentarios_posts`
6     FOREIGN KEY (`posts_id`)
7     REFERENCES `1_platziblog_db`.`posts` (`id`)
8     ON DELETE CASCADE
9     ON UPDATE CASCADE;
10
```

MySQL Workbench

Local instance MySQL57

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

1\_platziblog\_db

Tables

categorias

comentarios

etiquetas

posts

usuarios

Views

Stored Procedures

Functions

sys

Query 1 categorias - Table etiquetas - Table usuarios - Table posts - Table comentarios - Table

Table Name: comentarios Schema: 1\_platziblog\_db Charset/Collation: utf8 utf8\_bin Engine: InnoDB

Comments:

Foreign Key Name Referenced Table Column Referenced Column Foreign Key Options

comentarios\_posts '1\_platziblog\_db'.post id

comentarios\_usuario '1\_platziblog\_db'.usuarios usuarios\_id id

On Update: CASCADE On Delete: CASCADE

Skip in SQL generation

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Administration Schemas

Information

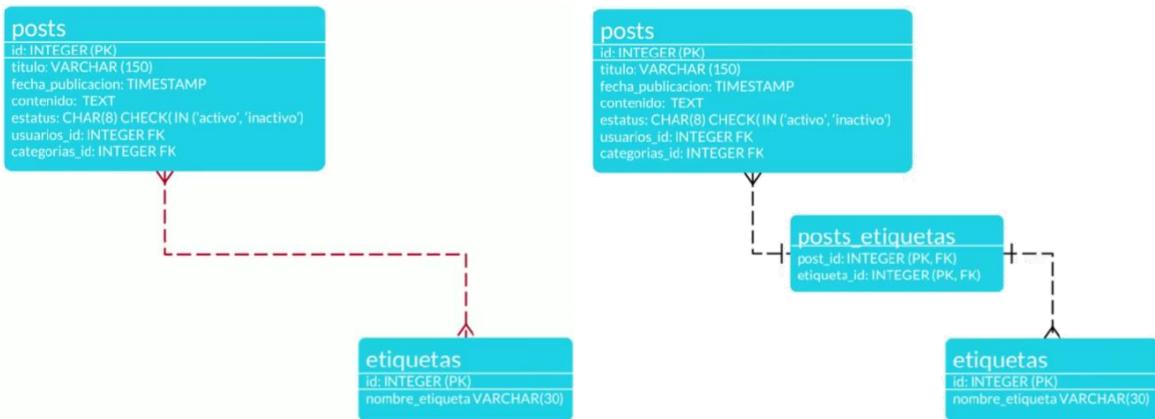
Schema: 1\_platziblog\_db

Columns Indexes Foreign Keys Triggers Partitioning Options

Apply Revert

Context Help Snippets

3. Finalmente nos haremos cargo de las **relaciones internas** con **cardinalidad N:N**, donde recordemos que se crea una **tabla transitiva** que cree **identificadores únicos** para su **conexión**.



Para ello primero se crea la tabla normal de etiquetas.

Online DDL

Algorithm: Default Lock Type: Default

```

1  CREATE TABLE `1_platziblog_db`.`etiquetas` (
2      `id` INT NOT NULL AUTO_INCREMENT,
3      `nombre_etiqueta` VARCHAR(30) NOT NULL,
4      PRIMARY KEY (`id`));
5

```

Y después se crea la tabla interna transitiva, que no almacena datos propios, sino que solamente crea identificadores únicos que denoten la conexión con cardinalidad N:N, donde de nuevo se deberán declarar sus llaves foráneas.

Online DDL

Algorithm: Default Lock Type: Default

```

1  CREATE TABLE `1_platziblog_db`.`posts_etiquetas` (
2      `id` INT NOT NULL AUTO_INCREMENT,
3      `posts_id` INT NOT NULL,
4      `etiquetas_id` INT NOT NULL,
5      PRIMARY KEY (`id`));
6

```

Online DDL

Algorithm: Default Lock Type: Default

```

1  ALTER TABLE `1_platziblog_db`.`posts_etiquetas`
2  ADD INDEX `postsetiquetas_posts_idx` (`posts_id` ASC);
3  ;
4  ALTER TABLE `1_platziblog_db`.`posts_etiquetas`
5  ADD CONSTRAINT `postsetiquetas_posts`
6  FOREIGN KEY (`posts_id`)
7  REFERENCES `1_platziblog_db`.`posts` (`id`)
8  ON DELETE CASCADE
9  ON UPDATE CASCADE;
10

```

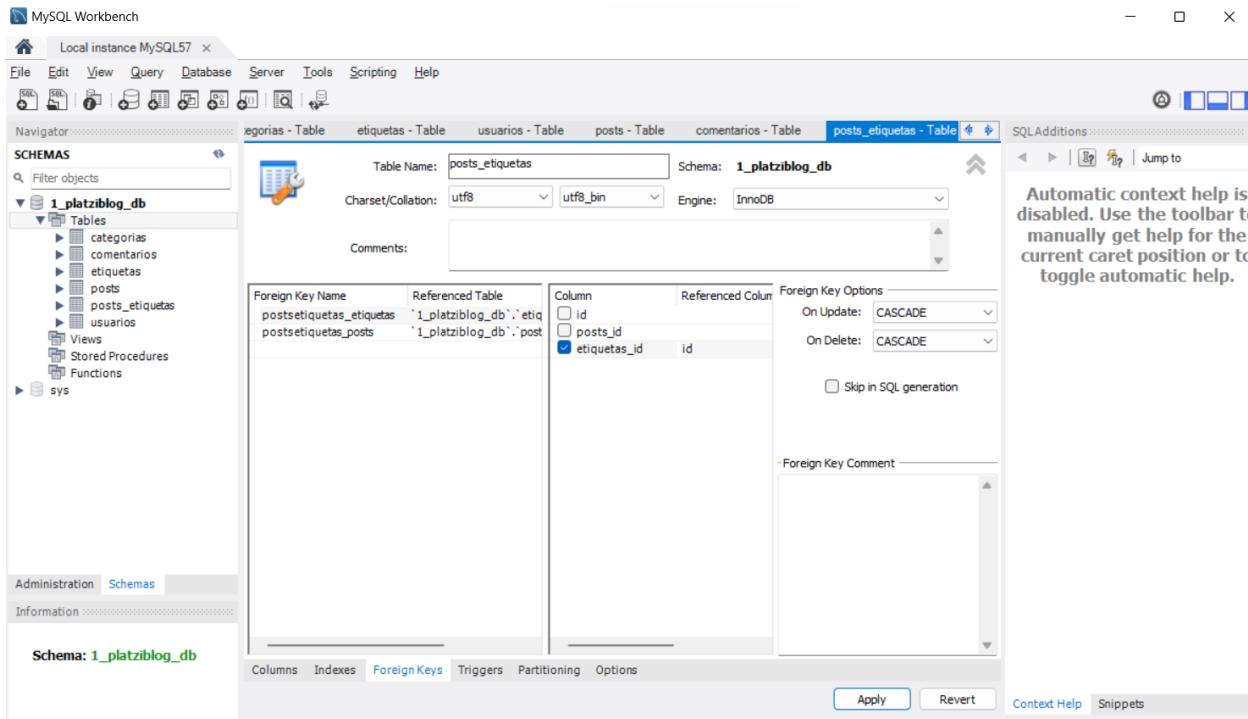
Online DDL

Algorithm: Default Lock Type: Default

```

1  ALTER TABLE `1_platziblog_db`.`posts_etiquetas`
2  ADD INDEX `postsetiquetas_etiquetas_idx` (`etiquetas_id` ASC);
3  ;
4  ALTER TABLE `1_platziblog_db`.`posts_etiquetas`
5  ADD CONSTRAINT `postsetiquetas_etiquetas`
6  FOREIGN KEY (`etiquetas_id`)
7  REFERENCES `1_platziblog_db`.`etiquetas` (`id`)
8  ON DELETE CASCADE
9  ON UPDATE CASCADE;

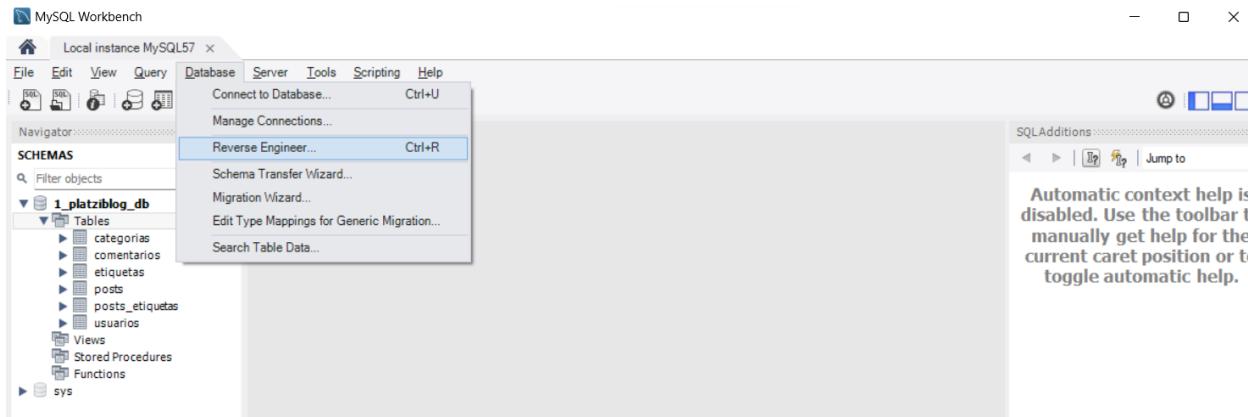
```



## Base de Datos con Código SQL a Diagrama Físico en MySQL Workbench

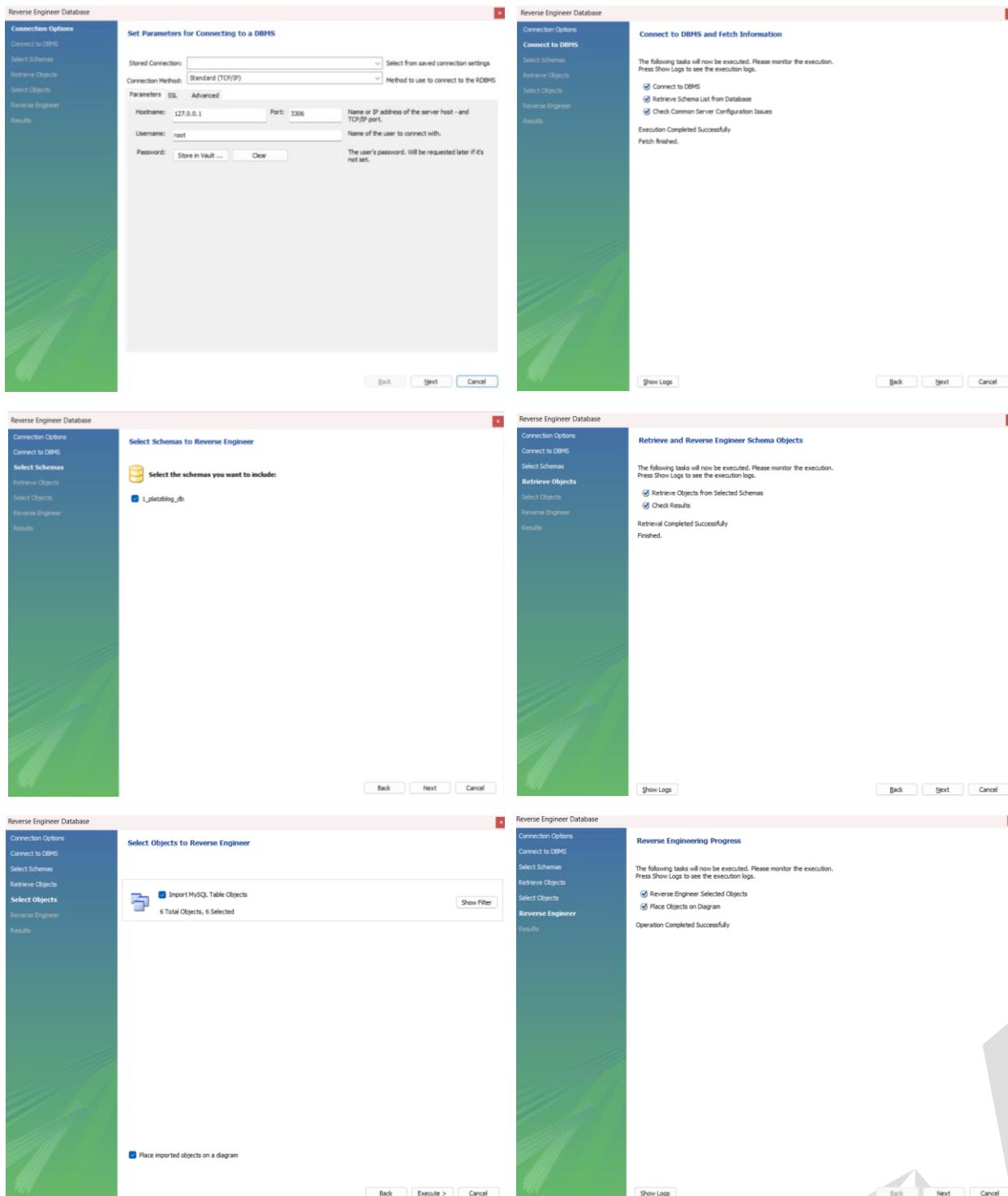
Finalmente, para observar el resultado de la codificación de nuestra base de datos en un diagrama físico dentro del cliente gráfico MySQL Workbench lo que haremos será lo siguiente:

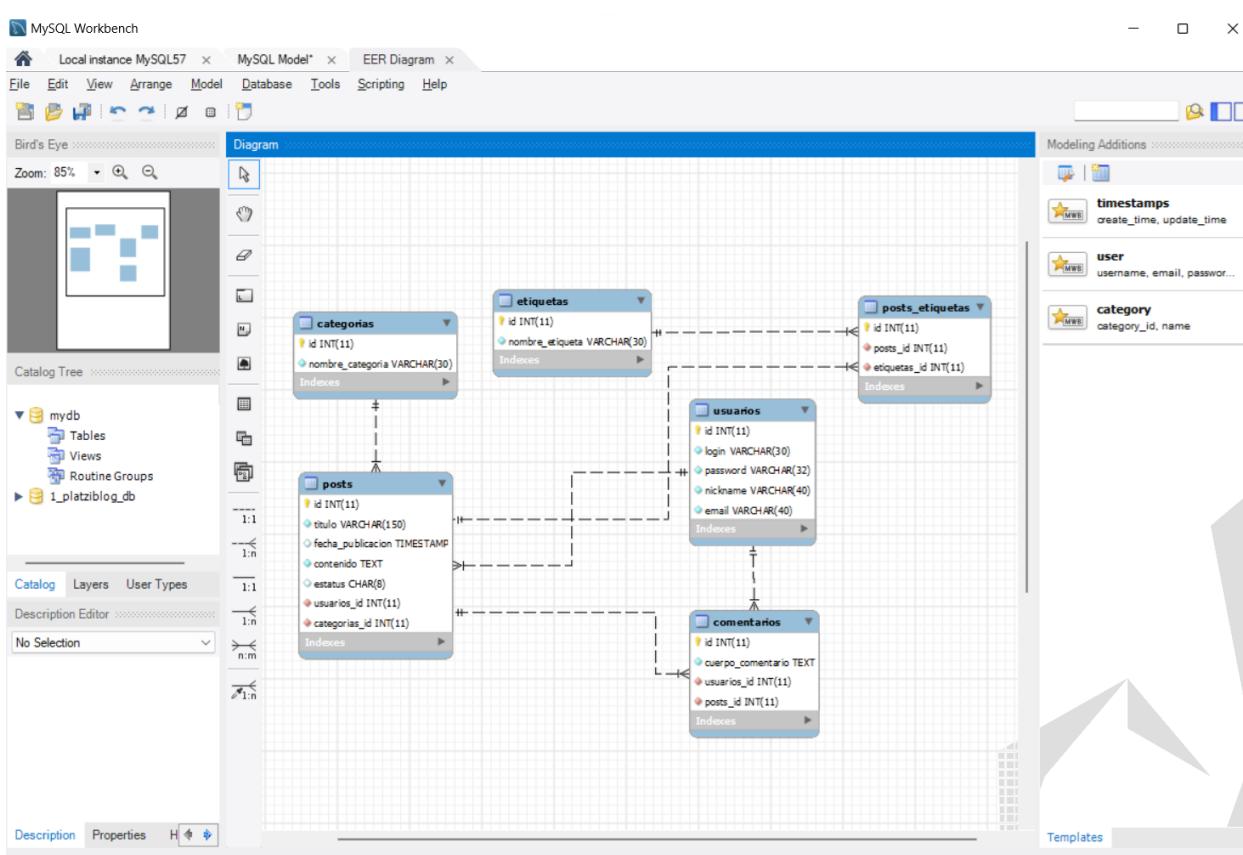
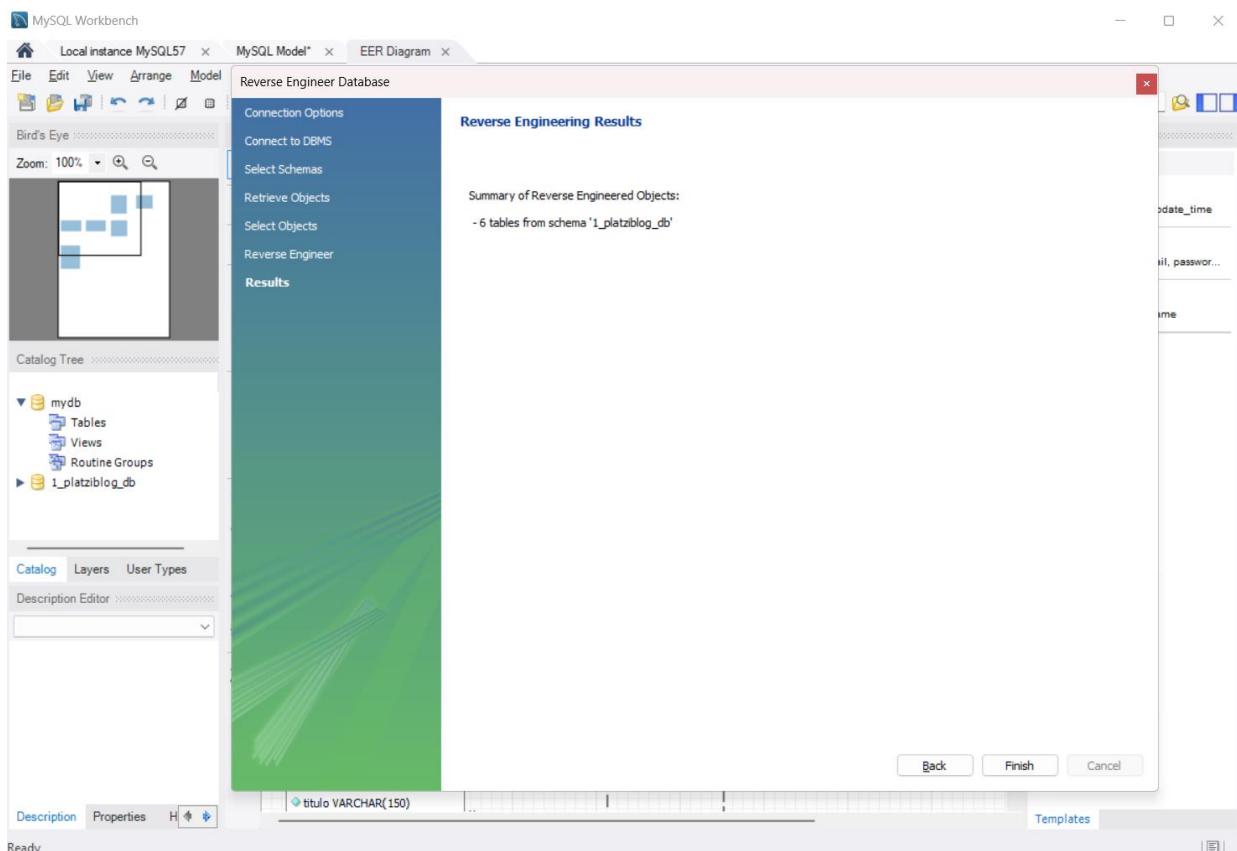
4. Daremos clic en la opción de Database → Reverse Engineer...



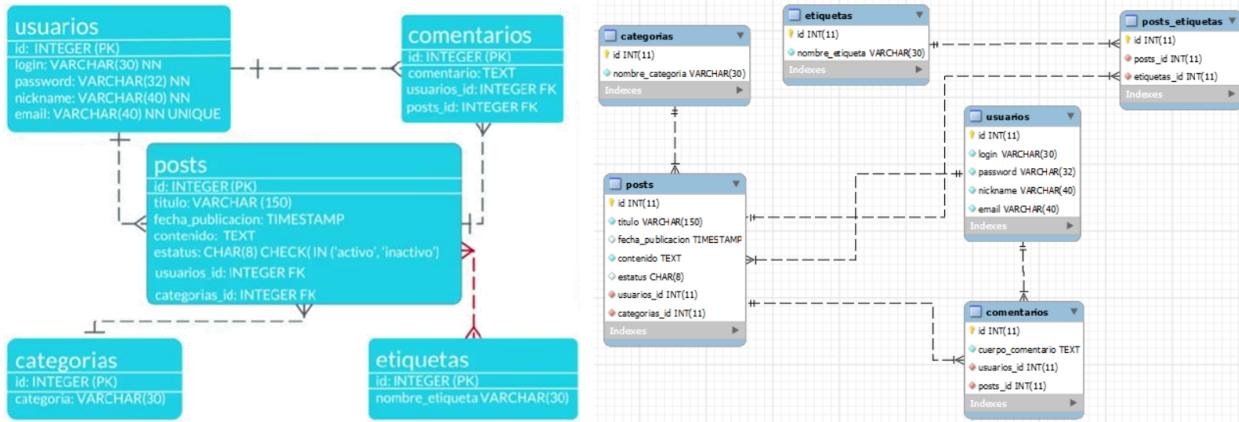
5. Daremos acceso al puerto donde se encuentra localizada la base de datos → Next → Introduciremos nuestra contraseña → Next → Seleccionaremos la base de datos de la cual queremos obtener el Diagrama Físico → Next → Daremos permiso para que importe todos los objetos de la base de datos (**Entidades, Atributos, Relaciones, Tipos de Datos, etc.**) → Next → Y finalmente se me indicará que se han importado todas las tablas de la base de datos, aquí es

donde ya podremos visualizar el diagrama físico obtenido como resultado del montaje de la base de datos en MySQL Workbench.





6. De esta manera podemos observar que la conexión de la base de datos fue correcta.



## Consultas a Bases de Datos: Query - SELECT

Las consultas o Queries son una parte fundamental de las bases de datos, ya que de esta forma es como se pueden extraer los datos de la db para realizar un análisis, responder una pregunta o simplemente utilizar dicha información, como se realiza con las aplicaciones de business intelligence, machine learning, data science, etc.

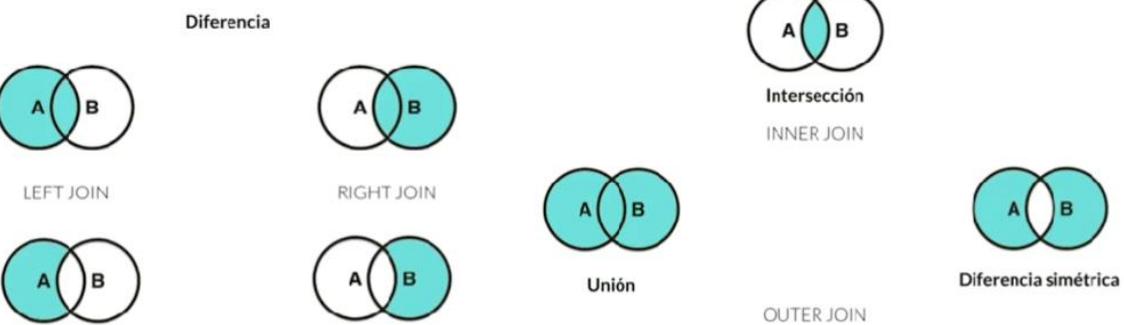
La estructura básica de un Query se conforma de los comandos **SELECT**, **FROM** y **WHERE** para indicar la posición y el elemento de donde se busca obtener cierta información. La **tabla** de la cual se buscan extraer los datos se indica con el comando **FROM**, la **columna (atributo)** se indica con el comando **SELECT** y la fila se señala con el comando **WHERE**.

- Para las consultas se utiliza el comando **SELECT**, en el cual se indica el nombre del **atributo** o **columna** de datos que se quiere extraer y a qué **tabla** pertenecen por medio de la instrucción **FROM**. Si después de la instrucción se utiliza un asterisco \* en vez del nombre de una **columna**, es porque se quieren extraer todos los datos de dicha **entidad**.
  - Existe una instrucción adicional que se puede utilizar en conjunto con el comando **SELECT** o **FROM** y se llama **AS**, la cual sirve para cambiar el nombre de la columna o tabla de datos extraída y asignarle un alias o nombre de variable, cambiando solo la forma en la que se presentan los datos, no en sí el nombre en la base de datos.
  - De igual forma existe el método **COUNT()** que siempre se debe utilizar alado del método **SELECT**, el cual recibe como parámetro un **atributo** de datos pertenecientes a la **tabla** y retorna el número de filas de datos que pertenecen a dicha **columna**.
  - El método **SUM()** sirve para sumar todos los valores numéricos de la tabla.
  - La función **GROUP\_CONCAT()** obtiene todas las instancias de una tupla de texto pertenecientes a un **atributo** separadas por comas.
  - Se había mencionado previamente que a través de la sentencia **FROM** se indica de qué **tabla** o **entidad** se extraerán los datos requeridos, aunque solo se estableció el caso donde esto se realizaba para una sola **entidad**, pero cuando se quiera extraer datos de **dos tablas distintas**, se añade la instrucción **JOIN**, pero es muy importante

mencionar que esto solo se podrá realizar en aquellas **entidades** que se encuentren enlazadas a través de una **relación**, osea cuando una contenga una **PRIMARY KEY** y la otra posea una **FOREIGN KEY**, ambas unidas a través de un **índice** en el diagrama físico.

- Se puede representar de forma gráfica el funcionamiento de una instrucción **JOIN** a través de un **Diagrama de Venn**.
  - Los pasos para **relacionar** los datos de ambas **tablas** son los siguientes:
    - Primero se indica a través del método **FROM** la **entidad** que tenga una **cardinalidad** de **1** (la cual adoptará la posición izquierda en el **Diagrama de Venn**).
    - Luego a través de alguna variante de **Diferencia**, **Intersección**, **Unión** o **Diferencia Simétrica** del método **JOIN** se denota la **entidad** con **cardinalidad** de **N** (que tomará la posición derecha).
    - Finalmente, ambas se **conectan** a través de la instrucción **ON** que se acompaña tanto del **atributo** que representa el **PRIMARY\_KEY** en la **tabla izquierda** como del **atributo** que represente el **FOREIGN\_KEY** de la **entidad derecha** y ambos se igualan.

## JOIN



- Además de forma opcional se podrá indicar exactamente a cuáles **filas** de la **tabla** nos estamos refiriendo, filtrándola a través de cierta **condición lógica** (**=, >, <, etc.**), ya que la extracción se puede realizar en una o varias filas, para ello se utiliza el comando **WHERE** acompañado del **valor** de algún **atributo**.
  - Si se quiere agregar más de un filtro en una búsqueda, lo que se hace es agregar después del primer filtro la sentencia **AND** y con eso se podrán sumar filtros adicionales.
- La sentencia **GROUP BY** de igual forma es opcional y sirve para ordenar la información obtenida de la consulta en forma de filas que agruparán los datos de una forma deseada, esto se logra al aplicar la sentencia a alguna de las **dos columnas** que hayan sido declaradas previamente en el comando **SELECT** y su resultado se verá reflejado en la **clasificación de la información** en función del **atributo** mencionado.
  - Normalmente este tipo de instrucción se declara en conjunto con el método **COUNT()** o **SUM()** dentro de los **dos atributos** a los que se les aplica el método **SELECT**, para que de esta manera se haga un **conteo de los datos** de la **columna** a la que no se le **está aplicando el método de conteo** y de esta forma se **agrupen y cuenten** los elementos conforman a cada clasificación de dicho **atributo** de la **tabla**.
  - La forma en la que se utiliza el método **GROUP BY** depende mucho de la información que contenga la base de datos, ya que a través de ella se podrán hacer informes agrupados por cierta clasificación, pero como podemos ver, existen ciertos algoritmos ya preestablecidos que sirven para obtener cada resultado.
  - **HAVING** igualmente se usa de forma opcional y lo que hace es **filtrar** a través de cierta **condición lógica** las **filas de información** extraídas de una **tabla**, de la misma forma cómo funciona el método **WHERE**, pero si hacemos pruebas con este, podremos ver que no funciona después de haber agrupado los datos obtenidos con el método **GROUP BY**, por lo que se debe reemplazar con la sentencia **HAVING** cuando se cumpla esta condición, pero realiza la misma función.
- El comando **ORDER BY** también es opcional y su función es la de ordenar una agrupación de datos para observar de mejor manera el resultado, cuando se busca que este orden se ejecute de forma **ascendente (de menos a más)** se incluye la sentencia **ASC** y cuando se quiere que se ordenen de forma **descendente (de más a menos)** se añade la sentencia **DESC**.
  - El comando **ORDER BY** se puede acompañar de la instrucción **LIMIT** la cual después de haber organizado los datos, limita el número de filas que se van a mostrar. Aunque esta sentencia se suele utilizar después del comando **ORDER BY**, se puede utilizar cuando sea.

```

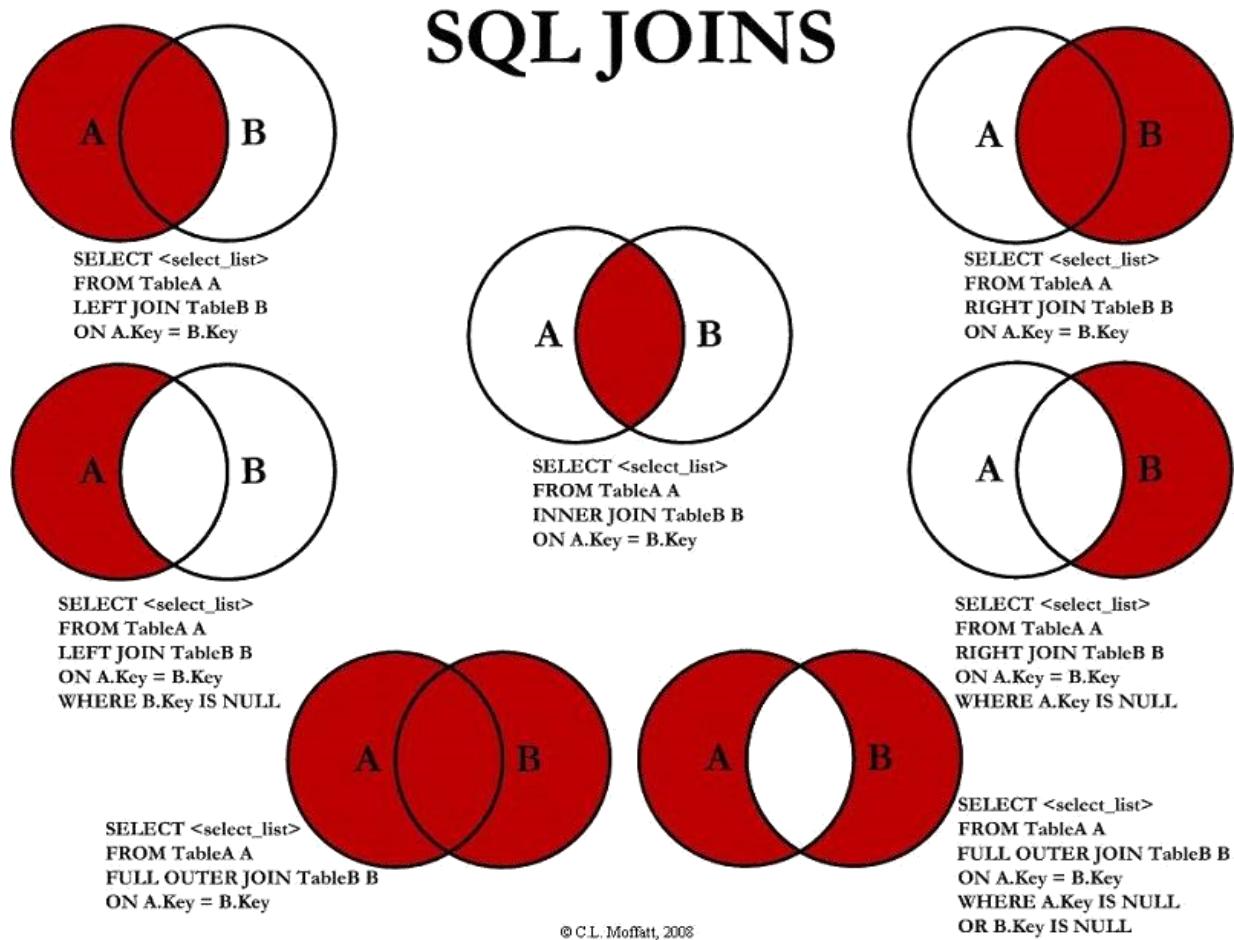
SELECT      Nombre_Columna_1 AS Nuevo_Nombre_Atributo_1, COUNT(Columna_n)
FROM        Nombre_Tabla_Izq
--Unión opcional de Diferencia, Intersección, etc. entre dos tablas diferentes.
JOIN        Entidad_Der ON Tabla_Izq.PRIMARY_KEY = Tabla_Der.FOREIGN_KEY
WHERE       Nombre_Atributo_o_Columna Operación Lógica "Valor_Fila_Para_Filtro"
AND         Nombre_Atributo Operación Lógica "Valor_Fila_Para_Filtro_Adicional"
  
```

```

GROUP BY Nombre_Columna_1
HAVING Nombre_Atributo_o_Columna Operación Lógica "Valor_Fila_Para_Filtro";
ORDER BY Nombre_Atributo_o_Columna ASC_o_DESC
LIMIT Número_de_Filas_Ordenadas_a_Mostrar

```

A continuación, se mostrarán ejemplos de cómo realizar distintos JOIN utilizando SQL con dos conjuntos (entidades) de datos distintas.



## Nested Queries: Consultas Anidadas (Agujero de Conejo)

Una Query anidada se da cuando dentro de una consulta se introduce otra, esto es muy utilizado cuando dentro de alguna condición se quiere utilizar algún un **valor máximo o mínimo** perteneciente a la **columna (atributo)** de una **tabla (entidad)**, por lo que muchas veces se utiliza en conjunto con los métodos **MIN()** o **MAX()**, pero el gran problema que tiene es cuando esta búsqueda se va a realizar varias veces en una base de datos, ya que el tiempo de ejecución se incrementa exponencialmente, por esa razón es que hay que analizar detenidamente sus casos de uso para evitar así que se creen agujeros de conejo interminables. La sintaxis que se puede utilizar para ejecutar es la siguiente:

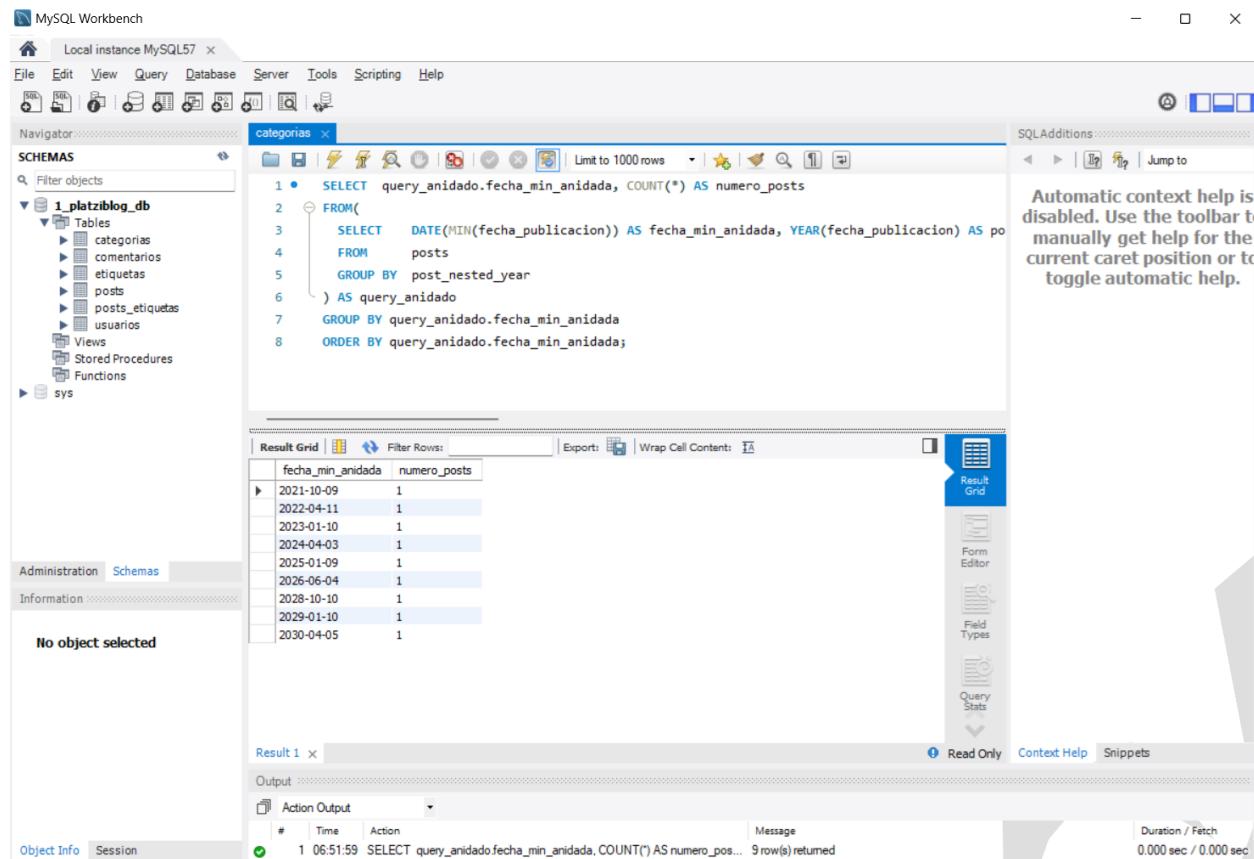
```

SELECT      Query_Anidado_1.Atributo_Anidado_1, COUNT(Columna)
FROM        (
--Consulta (Query) anidado.

SELECT      MIN(Atributo_1) AS Atributo_Anidado_1, COUNT(Columna_n)
FROM        Nombre_Tabla_o_Entidad
...
) AS Query_Anidado_1
GROUP BY    Query_Anidado_1.Atributo_Anidado_1
HAVING     Query_Anidado_1.Columna_n Operación Lógica "Valor_Fila_Para_Filtro";
ORDER BY    Query_Anidado_1.Atributo_Anidado_1 ASC_o_DESC
LIMIT      Número_de_Filas_Ordenadas_a_Mostrar

```

Hay que tener mucho cuidado con las consultas anidadas, pero no se puede negar su utilidad, ya que permiten primero hacer un análisis de la base de datos y luego hacer un análisis posterior con dicho resultado:



The screenshot shows the MySQL Workbench interface with a query editor window open. The query is a nested SELECT statement. The outer query selects from the inner query, which itself is a complex query involving multiple sub-selects and GROUP BY clauses. The result grid shows a single column named 'fecha\_min\_anidada' with values ranging from 2021-10-09 to 2030-04-05, each corresponding to a count of 1. The status bar at the bottom indicates the query took 0.000 sec / 0.000 sec.

```

1 •  SELECT query_anidado.fecha_min_anidada, COUNT(*) AS numero_posts
2   FROM(
3     SELECT DATE(MIN(fecha_publicacion)) AS fecha_min_anidada, YEAR(fecha_publicacion) AS po
4     FROM posts
5     GROUP BY post_nested_year
6   ) AS query_anidado
7   GROUP BY query_anidado.fecha_min_anidada
8   ORDER BY query_anidado.fecha_min_anidada;

```

fecha_min_anidada	numero_posts
2021-10-09	1
2022-04-11	1
2023-01-10	1
2024-04-03	1
2025-01-09	1
2026-06-04	1
2028-10-10	1
2029-01-10	1
2030-04-05	1

Otra aplicación de las consultas aplicadas es la siguiente, donde ahora el query interior fue hecho para obtener la condición que extrae solo cierta fila de la tabla:

```
SELECT      Nombre_Columna_1 AS Nuevo_Nombre_Atributo_1, COUNT(Columna_n)
FROM        Nombre_Tabla_o_Entidad
WHERE       Nombre_Atributo_o_Columna Operación Lógica (
--Consulta (Query) anidado.

SELECT      MAX(Atributo_1)
FROM        Nombre_Tabla_o_Entidad
...
)
```

Transformar una Pregunta en un Query de SQL - Ejemplos Prácticos

## De pregunta a Query

Lo que quieres mostrar = SELECT

De donde voy a tomar los datos = FROM

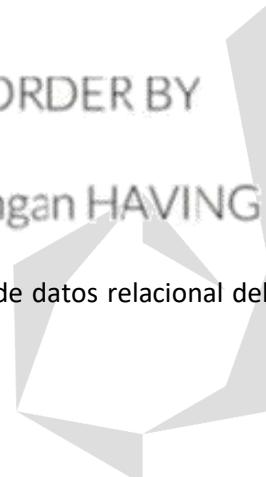
Los filtros de los datos que quieres mostrar = WHERE

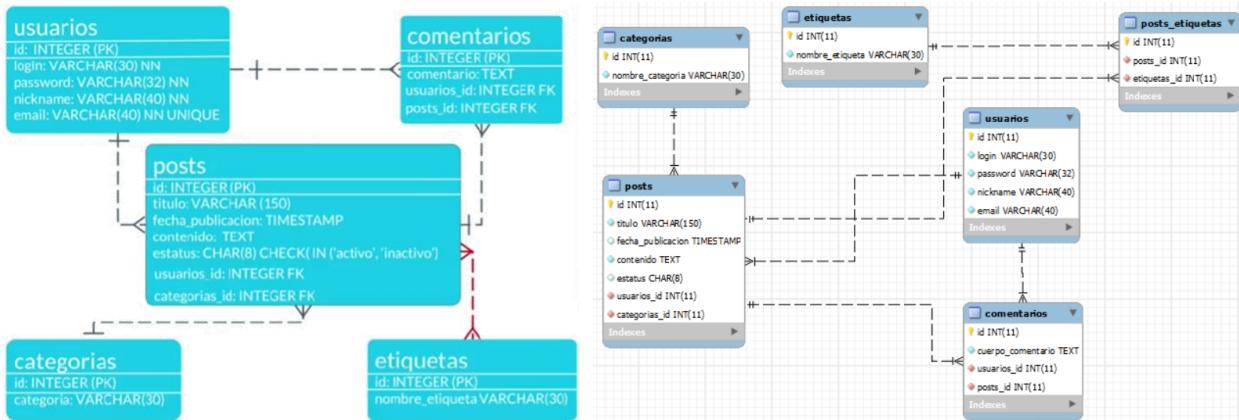
Los rubros por los que me interesa agrupar la información =  
GROUP BY

El orden en que quiero presentar mi información ORDER BY

Los filtros que quiero que mis datos agrupados tengan HAVING

A continuación, responderemos algunas preguntas de prueba acerca de la base de datos relacional del blog:





1. ¿Cuántas etiquetas tiene cada post del blog?

- Para identificar cada post podemos utilizar su título.
- La información proviene de 3 tablas distintas: posts, etiquetas y posts\_etiquetas (tabla intermedia por la cardinalidad N:N).
  - Debido a esta situación se deberá ejecutar un INNER JOIN doble que considere la intersección de las 3 tablas.
- La información se agrupa a través del id del post, ya que esa es la relación que hay entre la tabla de posts y la tabla de etiquetas y la información que quiero saber son las etiquetas contra el título del post.
- Podría colocar un orden numérico descendente para observar de más a menos el número de etiquetas de cada post.

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the schema **1\_platziblog\_db** containing tables: categorias, comentarios, etiquetas, posts, posts\_etiquetas, and usuarios.
- SQL Editor:** Displays the query:
 

```

1 SELECT posts.titulo, COUNT(*) num_etiquetas
2 FROM posts
3   INNER JOIN posts_etiquetas ON posts.id = posts_etiquetas.posts_id
4   INNER JOIN etiquetas ON etiquetas.id = posts_etiquetas.etiquetas_id
5 GROUP BY posts.id
6 ORDER BY num_etiquetas DESC;
```
- Result Grid:** Shows the results of the query:
 

titulo	num_etiquetas
Los mejores vestidos en la alfombra roja	4
La bolsa cae estrepitosamente	4
Se mejora la conducción autónoma de vehículos	4
Bienes raíces más baratos que nunca	3
Fuca OS sacude al mundo	3
Se descubre la unión entre astrofísica y física cu...	3
Químicos descubren nanomaterial	3
Equipo veterano da un gran espectáculo	3
Se descubre nueva partícula del modelo estandar	3
Escándalo en el mundo de la moda	3
Tenemos ganador de la fórmula e	2
Cierra campeonato mundial de football de mane...	2
Se fortalece el peso frente al dólar	2
U.S. Robotics presenta hallazgo	2
Ganador del premio Nobel por trabajo en genética	2
Tenemos un nuevo auto inteligente	2
Se presenta el nuevo teléfono móvil en evento	2
Tenemos campeona del mundial de voleibol	2
Ganan partido frente a visitantes	2
Los paparazzi captan escándalo en cámara	1
- Output:** Shows the execution log:
 

Action	Time	Action	Message	Duration / Fetch
SELECT * FROM posts WHERE fecha_publicacion = (	07:02:11	SELECT ...	1 row(s) returned	0.000 sec / 0.000 sec

2. Ahora que ya sé el número de etiquetas, ¿Cuáles etiquetas pertenecen a cada post del blog?
- Para saber el nombre de las etiquetas utilizaremos el método **GROUP\_CONCAT()** aplicado al **atributo** que indica el nombre de las etiquetas, manteniendo la misma estructura del código anterior.

The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays the schema **1\_platziblog\_db** with tables like **categorias**, **comentarios**, **etiquetas**, and **posts**. The central pane shows a query editor with the following SQL code:

```

1 • SELECT posts.titulo, GROUP_CONCAT(nombre_etiqueta) AS nombre_tag
2   FROM posts
3   INNER JOIN posts_etiquetas ON posts.id = posts_etiquetas.posts_id
4   INNER JOIN etiquetas ON etiquetas.id = posts_etiquetas.etiquetas_id
5   GROUP BY posts.id
6   ORDER BY nombre_tag ASC;
    
```

The Result Grid displays the output of the query, showing the title of each post and the concatenated list of its tags. The Output pane at the bottom shows the execution details: 21 row(s) returned in 0.000 sec / 0.000 sec.

3. ¿Existe alguna etiqueta que no corresponda a ningún post?
- Quiero mostrar todas las etiquetas que no estén ligadas a ningún post.
  - Los datos los voy a tomar de la tabla de las etiquetas, pero como quiero saber su conexión con post, no es necesario que analice post, solo la tabla de etiquetas y su tabla de transición intermedia.
  - Debido a esta situación se deberá ejecutar un LEFT JOIN (de la tabla etiquetas), doble se considere solo las etiquetas que no tengan conexión, osea A – B, siendo A = etiquetas y B = tabla\_intermedia\_con\_conexion\_a\_posts.
  - No es necesario agrupar la información.

The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays the schema **1\_platziblog\_db** with tables like **categorias**, **comentarios**, **etiquetas**, and **posts**. The central pane shows a query editor with the following SQL code:

```

1 • SELECT *
2   FROM etiquetas
3   LEFT JOIN posts_etiquetas ON etiquetas.id = posts_etiquetas.etiquetas_id
4   WHERE posts_etiquetas.etiquetas_id IS NULL;
    
```

The Result Grid displays the output of the query, showing a single row where the **posts\_id** column is null, indicating an orphaned tag.

4. ¿Cuál categoría posee el mayor número de posts?
- Los datos que se quieren mostrar son el nombre de la categoría y el número de posts que corresponden a cada una.
  - La información proviene de 2 tablas distintas: posts y etiquetas.
    - Como se busca encontrar los datos relacionados se deberá ejecutar un INNER JOIN que considere la intersección de las 2 tablas, osea  $A \cap B$ , recordemos que esto se logra al utilizar el índice que relaciona ambas tablas.
  - La información se agrupa a través del id de la categoría porque de esa manera se podrá mostrar cada tipo de categoría distinta.
  - Se colocará un orden numérico descendente para observar de mayor a menor el número de posts de cada etiqueta.

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database schema with the `1_platziblog_db` database selected, containing tables like `categorias`, `comentarios`, `etiquetas`, `posts`, `posts_etiquetas`, and `usuarios`.
- SQL Editor:** Displays the following SQL query:
 

```

1 •   SELECT c.nombre_categoria, COUNT(*) AS num_posts
2   FROM categorias AS c
3   INNER JOIN posts AS p ON c.id = p.categorias_id
4   GROUP BY c.id
5   ORDER BY num_posts DESC;
```
- Result Grid:** Shows the results of the query in a table:
 

nombre_categoria	num_posts
Deportes	5
Tecnología	5
Ciencia	4
Especiales	4
Economía	3
- SQLAdditions:** A note stating "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

5. ¿Qué usuario ha creado el mayor número de posts en el sistema?
- El procedimiento es el mismo al ejercicio anterior, pero cambiando la tabla de donde provienen los datos y el dato mismo que se quiere mostrar.
  - Opcionalmente nos podemos limitar a mostrar 1 solo valor, de esta manera mostrando solo el mayor o los primeros dos, para comprobar si en el segundo se repite el número.

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database schema with the `1_platziblog_db` database selected, containing tables like `categorias`, `comentarios`, `etiquetas`, `posts`, `posts_etiquetas`, and `usuarios`.
- SQL Editor:** Displays the following SQL query:
 

```

1 •   SELECT u.nickname, COUNT(*) AS num_posts
2   FROM usuarios AS u
3   INNER JOIN posts AS p ON u.id = p.usuarios_id
4   GROUP BY u.id
5   ORDER BY num_posts DESC
6   LIMIT 2;
```
- Result Grid:** Shows the results of the query in a table:
 

nickname	num_posts
Moni	9
Israel	6
- SQLAdditions:** A note stating "Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help."

6. ¿De qué categorías (temas) están escribiendo los 3 usuarios que han creado el mayor número de posts en el sistema?

- El código resultante del ejercicio anterior se repite, pero se debe añadir el dato adicional que se está solicitando, que en este caso son las categorías de las que está escribiendo el usuario.
  - Para obtener y mostrar una lista de las categorías de temas de los que escribe cada usuario se utiliza el método **GROUP\_CONCAT()** aplicado al nombre de las categorías de temas.
- Como la información proviene de 3 tablas distintas: usuarios, categorías y posts, se debe realizar una interconexión de todas ellas.
  - Debido a que se están buscando los datos que pertenezcan a las 3 tablas a la vez, se deberá ejecutar un INNER JOIN doble que considere la intersección de las 3 tablas osea  $A \cap B \cap C$ , recordemos que esto se logra al utilizar el índice que relaciona cada una de las tablas por separado.

```

SELECT u.nickname, COUNT(*) AS num_posts, GROUP_CONCAT(c.nombre_categoria)
FROM usuarios AS u
INNER JOIN posts AS p ON u.id = p.usuarios_id
INNER JOIN categorias AS c ON c.id = p.categorias_id
GROUP BY u.id
ORDER BY num_posts DESC
LIMIT 3;
  
```

nickname	num_posts	GROUP_CONCAT(c.nombre_categoria)
Moni	9	Deportes,Deportes,Deportes,Tecnología,Ciencia,Ciencia,Economía,Economía,Deportes
Israel	6	Tecnología,Tecnología,Tecnología,Tecnología,Economía,Deportes
Ed	4	Espectáculos,Espectáculos,Espectáculos,Espectáculos

7. ¿Qué usuarios no han escrito ningún post?

- Se busca mostrar todos los nombres de los usuarios que no estén ligados a ningún post.
- Los datos los voy a tomar de la tabla de los usuarios y de los posts.
  - Debido a que quiero saber todos los usuarios que no tengan ningún post se ejecutará una operación de LEFT JOIN, doble se considere solo los usuarios que no tengan conexión, osea  $A - B$ , siendo  $A =$ usuarios y  $B =$ posts.
  - El filtro que se aplicará es encontrar las filas de datos donde el post sea nulo para lograr la operación  $A - B$ .

```

SELECT *
FROM usuarios AS u
LEFT JOIN posts AS p ON u.id = p.usuarios_id
WHERE p.usuarios_id IS NULL;
  
```

	id	login	password	nickname	email	id	título	fecha_publicacion	conte
5	5	perezoso	&N_<JS_Y)*(&TG...	Oso Pérez	perezoso@platziblog.com	HULL	HULL	HULL	HULL

## Basic Queries

```
-- filter your columns  
SELECT col1, col2, col3, ... FROM table1  
-- filter the rows  
WHERE col4 = 1 AND col5 = 2  
-- aggregate the data  
GROUP by ...  
-- limit aggregated data  
HAVING count(*) > 1  
-- order of the results  
ORDER BY col2
```

### Useful keywords for SELECTS:

**DISTINCT** - return unique results  
**BETWEEN a AND b** - limit the range, the values can be numbers, text, or dates  
**LIKE** - pattern search within the column text  
**IN (a, b, c)** - check if the value is contained among given.

## Data Modification

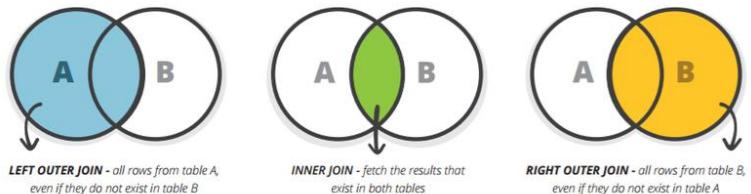
```
-- update specific data with the WHERE clause  
UPDATE table1 SET col1 = 1 WHERE col2 = 2  
-- insert values manually  
INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)  
VALUES (1, 'Rebel', 'Labs');  
-- or by using the results of a query  
INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)  
SELECT id, last_name, first_name FROM table2
```

## Views

A **VIEW** is a virtual table, which is a result of a query. They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS  
SELECT col1, col2  
FROM table1  
WHERE ...
```

## The Joy of JOINS



**LEFT OUTER JOIN** - all rows from table A, even if they do not exist in table B

**INNER JOIN** - fetch the results that exist in both tables

**RIGHT OUTER JOIN** - all rows from table B, even if they do not exist in table A

## Updates on JOINed Queries

You can use **JOINS** in your **UPDATES**  
UPDATE t1 SET a = 1  
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1\_id  
WHERE t1.col1 = 0 AND t2.col2 IS NULL;

NB! Use database specific syntax, it might be faster!

## Semi JOINS

You can use subqueries instead of **JOINS**:

```
SELECT col1, col2 FROM table1 WHERE id IN  
(SELECT t1_id FROM table2 WHERE date >  
CURRENT_TIMESTAMP)
```

## Indexes

If you query by a column, index it!  
CREATE INDEX index1 ON table1 (col1)

### Don't forget:

Avoid overlapping indexes  
Avoid indexing on too many columns  
Indexes can speed up **DELETE** and **UPDATE** operations

## Useful Utility Functions

-- convert strings to dates:  
**TO\_DATE** (Oracle, PostgreSQL), **STR\_TO\_DATE** (MySQL)

-- return the first non-NULL argument:

**COALESCE** (col1, col2, "default value")

-- return current time:

**CURRENT\_TIMESTAMP**

-- compute set operations on two result sets

**SELECT** col1, col2 **FROM** table1

**UNION / EXCEPT / INTERSECT**

**SELECT** col3, col4 **FROM** table2;

**Union** - returns data from both queries

**Except** - rows from the first query that are not present in the second query

**Intersect** - rows that are returned from both queries

## Reporting

Use aggregation functions

**COUNT** - return the number of rows

**SUM** - cumulate the values

**AVG** - return the average for the group

**MIN / MAX** - smallest / largest value

BROUGHT TO YOU BY  
**XRebel**

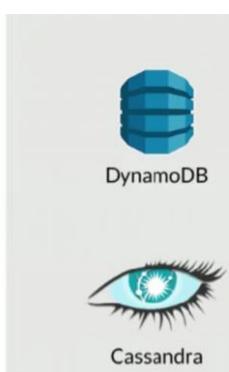
# Bases de Datos No Relacionales

Como ya se había mencionado previamente, las bases de datos no relacionales o NRDB (Non Relational Data Bases) no se conforman de un solo tipo de bases de datos, sino de varios, y aunque puedan ser muy distintas unas de otras, todas se engloban dentro de la misma categoría de base de datos no relacional que utilizan lenguajes NoSQL (Not Only SQL). Los diferentes tipos de bases de datos no relacionales son:

- **NRDB Basadas en Clave-Valor:** Estas bases de datos no relacionales están hechas con el objetivo de guardar y extraer datos de forma rápida, todo a través de una clave. La particularidad que tienen es que varios datos están ligados a un mismo id y para hacer consultas dentro de ese grupo de datos se debe utilizar una clave adicional llamada hash o hashmap.
  - Algunos ejemplos de estas bases de datos no relacionales son DynamoDB de AWS y Cassandra de Facebook.

## Clave - valor

Son ideales para almacenar y extraer datos con una clave única. Manejan los diccionarios de manera excepcional.



- **NRDB Basadas en Grafos:** Los grafos se componen de nodos o entidades (tablas) que tienen relaciones muy complejas con otras y generalmente se relacionan todos contra todos, su mayor uso es en el de la creación de inteligencia artificial o redes neuronales.

## Basadas en grafos

Basadas en teoría de grafos sirven para entidades que se encuentran interconectadas por múltiples relaciones. Ideales para almacenar relaciones complejas.



- **NRDB Basadas en Memoria:** Este tipo de base de datos son sumamente rápidas, pero tienen la gran desventaja de que son volátiles.

## En memoria

Pueden ser de estructura variada, pero su ventaja radica en la velocidad, ya que al vivir en memoria la extracción de datos es casi inmediata.



Memcached



Redis

- **NRDB Optimizadas para Búsquedas:** Este tipo de base de datos pueden ejecutar queries muy complejos de forma muy rápida a grandes repositorios de datos históricos que almacenan un gran volumen de datos, son muy utilizadas en aplicaciones de business intelligence y machine learning.

- Algunos ejemplos de estas bases de datos no relacionales son BigQuery de Google y Elasticsearch.

## Optimizadas para búsqueda

Pueden ser de diversas estructuras, su ventaja radica en que se pueden hacer queries y búsquedas complejas de manera sencilla.



BigQuery

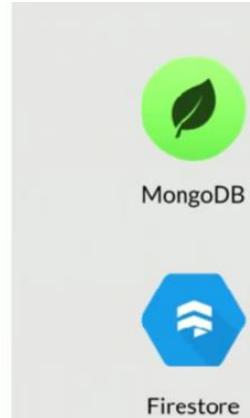


Elasticsearch

- **NRDB Basadas en Documentos:** El concepto de Documento es mayormente utilizado para referirnos a archivos de tipo JSON (JavaScript Object Notation) o XML.
  - Algunos ejemplos de estas bases de datos no relacionales son MongoDB y FireStore de Google.

## Basados en documentos

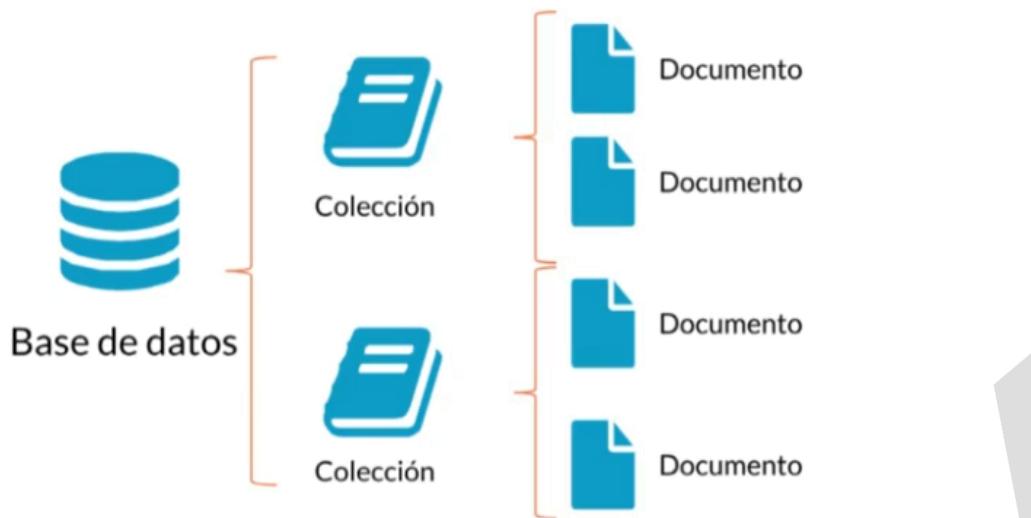
Son una implementación de clave valor que varía en la forma semiestructurada en que se trata la información. Ideal para almacenar datos JSON y XML.



## Jerarquía de Datos de las Bases de Datos No Relacionales: FireStore

En las bases de datos no relacionales, en vez de que se cuente con tablas (entidades), atributos (columnas), relaciones (conexiones), etc. su estructura se basa en colecciones de datos, que son el equivalente a las entidades, las cuales clasifican los distintos documentos que contienen estructuras JSON que asocian cada valor con una clave.

## Jerarquía de datos en firestore



Cuando trabajamos con bases de datos basadas en documentos como FireStore, cambiaremos el concepto de las tablas por las colecciones y las tuplas (filas) por los documentos.

- **Tabla** → Colección.
- **Tupla** → Documento.

Además, en el contexto de las colecciones, identificamos dos categorías principales:

- Las "Top level collections" o colecciones de nivel superior.
- Y las "subcollections" o subcolecciones, que se incorporan dentro de otra colección.

No existe una regla estricta para determinar si la colección debe ser de nivel superior o una subcolección al crear una base de datos basada en documentos; más bien, esta decisión depende del caso de uso específico.

Una consideración clave al diseñar la base de datos es anticipar cómo se extraerán los datos. En el contexto de una aplicación, es útil pensar en términos de las vistas que se mostrarán en un momento específico. En otras palabras, al estructurar la base de datos, debemos asegurarnos de que refleje o contenga, al menos, todos los datos necesarios para satisfacer los requisitos visuales de nuestra aplicación en un momento dado.

Esta regla se aplica salvo algunas excepciones, como cuando se tiene una entidad que necesita existir y modificarse de manera constante e independiente de otras colecciones.

## Conclusiones y Aplicaciones

Al final, no se debe elegir una sola base de datos para que funcione con un proyecto en específico, sino que, conociendo el funcionamiento de cada tipo, se pueden utilizar distintas bases de datos que tengan propósitos específicos, como realizar reportes o análisis de datos con SQL o permitir un ingreso y extracción de datos rápida con NoSQL.

Big Data

Data Warehouse

Data Mining

ETL



Business Intelligence

Machine Learning

Data Science

## Referencias

Platzi, Israel Vázquez, "Curso de Fundamentos de Bases de Datos", 2018 [Online], Available: <https://platzi.com/new-home/clases/1566-bd/19781-bienvenida-conceptos-basicos-y-contexto-historico-/>

