

Fundamentos de Bases de Datos

Modulo 1. Bienvenida conceptos básicos y contexto histórico de las Bases de Datos

Clase 1 Expert Session: resuelve tus dudas sobre las bases de datos

Clase 2 Bienvenida conceptos básicos y contexto histórico de las Bases de Datos

El almacenamiento en la nube tiene un gran pro comparada con los otros métodos de almacenamiento ya que es accesible desde cualquier parte del mundo. Además es centralizada y puede ser usada por varias personas al mismo tiempo.

Las bases de datos entran cuando hacemos la transición a medios digitales.

Tipos de bases de datos

Relacionales: En la industria hay varias compañías dedicadas a ser manejadoras de bases de datos relacionales como SQL Server, Oracle, **MariaDB**, entre otras. **No relacionales:** Todavía están avanzando y existen ejemplos muy distintos como cassandra, elasticsearch, neo4j, MongoDB, entre otras.

Servicios

Auto administrados: Es la base de datos que instalas tú y te encargas de actualizaciones, mantenimiento, etc.

Administrados: Servicios que ofrecen las nubes modernas como Azure y no debes preocuparte por mantenimiento o actualizaciones.

Modulo 2. Introducción a las bases de datos relacionales

Clase 3 Historia de las RDB (relational data bases)

Las bases de datos surgen de la necesidad de conservar la información más allá de lo que existe en la memoria RAM.

Las bases de datos basadas en archivos eran datos guardados en texto plano, fáciles de guardar pero muy difíciles de consultar y por la necesidad de mejorar esto nacen las bases de datos relacionales. **Su inventor Edgar Codd** dejó ciertas reglas para asegurarse de que toda la filosofía de las bases de datos no se perdiera, estandarizando el proceso, **Codd invento el álgebra relacional**

(Contenido adicional) Bases de datos relacionales (RBD)

Es importante que sea fácil de guardar y extraer, anteriormente se usaban bases de datos basadas en archivos, el cuál era texto plano fácil de guardar, pero difícil de extraer, por esto se inventaron las bases de datos relacionales. En 1990 Codd se preocupó porque los sistemas de gestión de bases de datos (SGBD) que decían ser relacionales, no lo eran. En la práctica es difícil cumplir las 12 pero, un SGBD es más relacional cuantas más reglas cumpla

Las Reglas y mandamientos de Edgar Frank Ted Codd

Regla 0: Regla de fundación. a) Cualquier sistema que se proclame como relacional, debe ser capaz de gestionar sus bases de datos enteramente mediante sus capacidades relacionales.

Regla 1: Regla de la información. a) Todos los datos deben estar almacenados en las tablas b) Esas tablas deben cumplir las premisas del modelo relacional c) No puede haber información a la que accedemos por otra vía

Regla 2: Regla del acceso garantizado. a) Cualquier dato es accesible sabiendo la clave de su fila y el nombre de su columna o atributo b) Si a un dato no podemos acceder de esta forma, no estamos usando un modelo relacional

Regla 3: Regla del tratamiento sistemático de valores nulos. a) Esos valores pueden dar significado a la columna que los contiene b) El SGBD debe tener la capacidad de manejar valores nulos c) El SGBD reconocerá este valor diferenciándolo de cualquier otro d) El SGBD deberá aplicársele la lógica apropiada e) Es un valor independiente del tipo de datos de la columna

Regla 4: Catálogo dinámico en línea basado en el modelo relacional. a) El catálogo en línea es el diccionario de datos b) El diccionario de datos se debe de poder consultar usando las mismas técnicas que para los datos c) Los metadatos, por tanto, se organizan también en tablas relacionales d) Si SELECT es una instrucción que consulta datos, también será la que consulta los metadatos

Regla 5: Regla comprensiva del sublenguaje de los datos completo. a) Al menos tiene que existir un lenguaje capaz de hacer todas las funciones del SGBD b) No puede haber funciones fuera de ese lenguaje c) Puede haber otros lenguajes en el SGBD para hacer ciertas tareas d) Pero esas tareas también se deben poder hacer con el "lenguaje completo"

Regla 6: Regla de actualización de vistas. a) Las vistas tienen que mostrar información actualizada b) No puede haber diferencias entre los datos de las vistas y los datos de las tablas base

Regla 7: Alto nivel de inserción, actualización, y cancelación. a) La idea es que el lenguaje que maneja la base de datos sea muy humano b) Eso implica que las operaciones del lenguaje de manipulación de los datos (DML) trabajen con conjuntos de filas a la vez c) Para modificar, eliminar o añadir datos, no hará falta programar de la forma en la que lo hacen los lenguajes de tercera generación como C o Java

Regla 8: Independencia física de los datos. a) Cambios en la física de la BD no afecta a las aplicaciones ni a los esquemas lógicos b) El acceso a las tablas (elemento lógico) no cambia porque la física de la base de datos cambie

Regla 9: Independencias lógicas de los datos. a) Cambios en el esquema lógico (tablas) de la BD no afectan al resto de esquemas b) Si cambiamos nombres de tabla, o de columna o modificamos información de las filas, las aplicaciones (esquema externo) no se ven afectadas c) Es más difícil de conseguir

Regla 10: Independencia de la integridad. a) Las reglas de integridad (restricciones) deben de ser gestionadas y almacenadas por el SGBD

Regla 11: Independencia de la distribución. a) Que la base de datos se almacene o gestione de forma distribuida en varios servidores, no afecta al uso de esta ni a la programación de las aplicaciones de usuario b) El esquema lógico es el mismo independientemente de si la BD es distribuida o no

Regla 12: La regla de la no subversión. a) La base de datos no permitirá que exista un lenguaje o forma de acceso, que permita saltarse las reglas anteriores.

Clase 4 Entidades y atributos

Una **entidad** es algo similar a un objeto (programación orientada a objetos) y representa algo en el mundo real, incluso algo abstracto. Tienen atributos que son las cosas que los hacen ser una entidad, se diagraman dentro de cuadrados y **por convención se ponen en plural**.

Atributos

Los atributos se representan con óvalos, los que tienen multiples valores llevan dobles óvalos, también existen atributos compuestos, los atributos especiales son óvalos con linea punteada.

Los **atributos compuestos** son aquellos que tienen atributos ellos mismos.

Los **atributos llave** son aquellos que identifican a la entidad y no pueden ser repetidos, se diagraman con underline. Existen:

- Naturales: Son inherentes al objeto como el número de serie
- Clave artificial: No es inherente al objeto y se asigna de manera arbitraria

Atributos



Entidades

Entidad Fuerte: No depende de ninguna entidad para existir

Entidades débiles: No pueden existir sin una entidad fuerte y se representan con un cuadrado con doble línea.

- Identidades débiles por identidad: No se diferencian entre sí más que por la clave de su entidad fuerte.
- Identidades débiles por existencia: Se les asigna una clave propia.

Entidades débiles

Libros

Ejemplares

Entidades débiles: identidad

Libros

id	título	...
LKJ789JKAS	Viaje al cent...	...
KCO3100KJH	El señor de
NSDJOIH128	De la tierra...	...
09KSIHBD71	Amor en tie...	...

Ejemplares

localización	edición
pasillo 1	1
pasillo 1	1
pasillo 1	3
pasillo 1	1

Las entidades débiles pueden serlo por dos motivos:

Por identidad: No se diferencian entre sí más que por la clave de su entidad fuerte (clave ISBN).

Por existencia: Usando llave asignada por nosotros (un ID).

Clase 5 Entidades de Platzi Blog

Nuestro proyecto será un manejador de Blogpost. Es un contexto familiar y nos representará retos muy interesantes.

- Primer paso: Identificar las entidades (abstracciones del mundo real).

Diagrama ER: Platziblog

usuarios

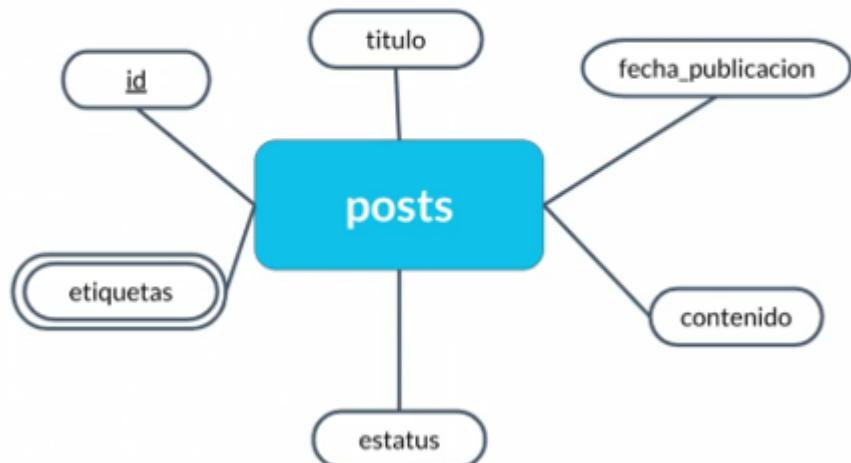
comentarios

posts

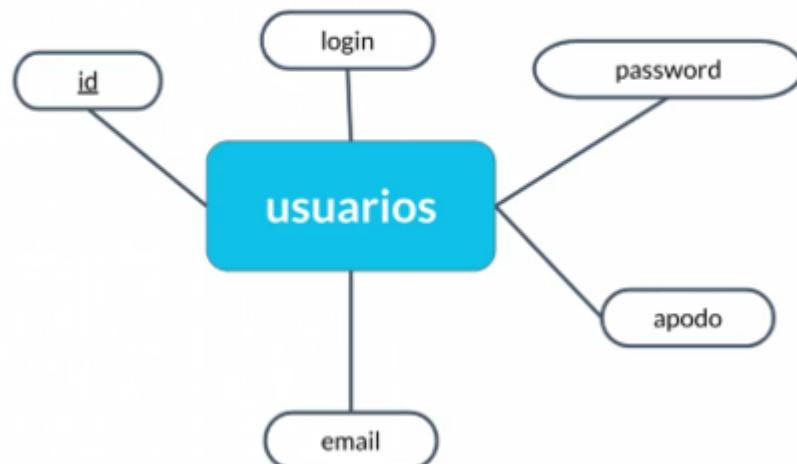
categorías

- Segundo paso: Pensar en los atributos.

Entidades Platziblog

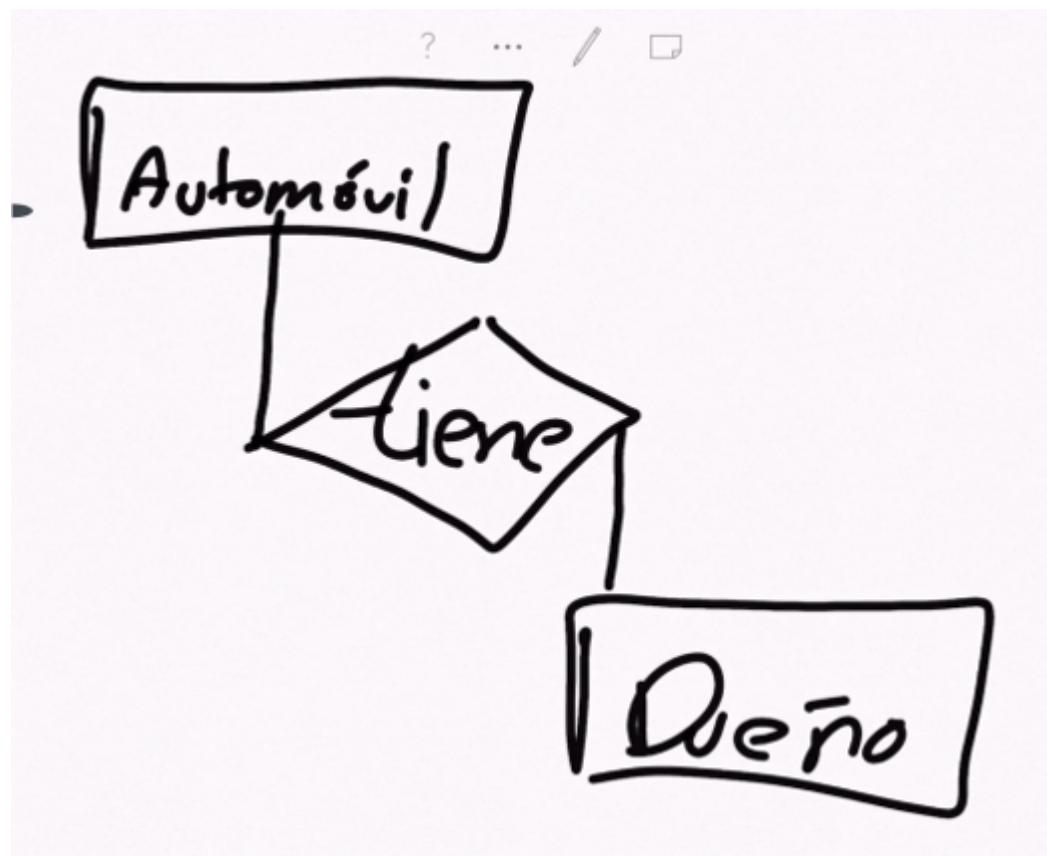


Entidades Platziblog



Clase 6 Relaciones

Las **relaciones** nos permiten ligar o unir nuestras diferentes entidades y se representan con rombos. Por convención se definen a través de verbos.

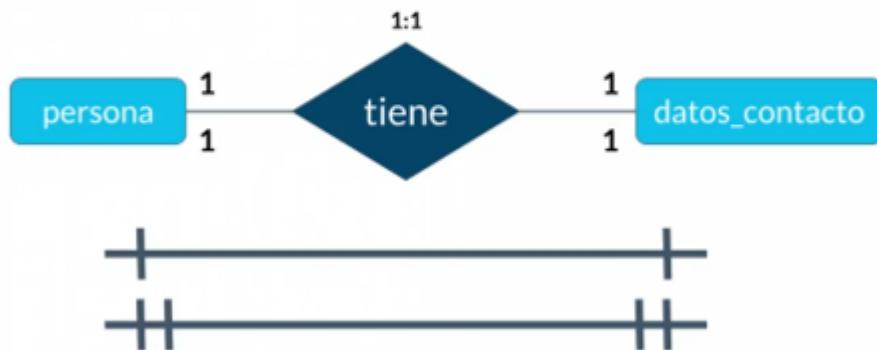


Las relaciones tienen una propiedad llamada **cardinalidad** y tiene que ver con números. Cuántos de un lado pertenecen a cuántos del otro lado:

Cardinalidad: 1 a 1 Cardinalidad: 0 a 1 Cardinalidad: 1 a N Cardinalidad: 0 a N

Cardinalidad: 1 a 1

Cardinalidad: 1 a 1

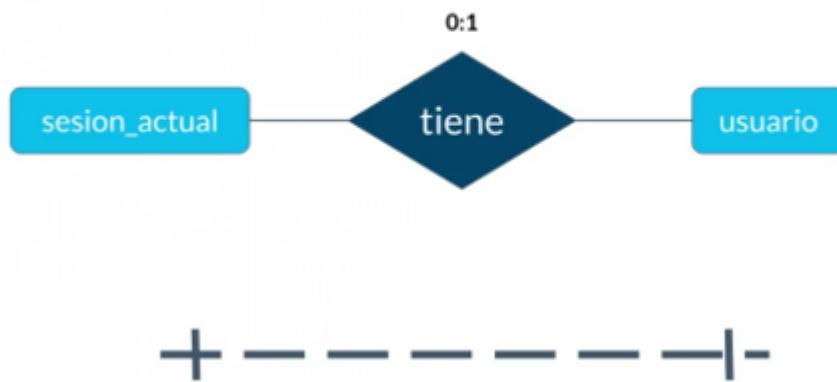


1 persona - tiene - 1 dato_contacto 1 dato_contacto - tiene - 1 persona.

Cardinalidad: 0 a 1

Algunos autores la denominan 1 a 1 opcional. En el ejemplo una sesión tiene un usuario, pero no necesariamente está loggeado todo el tiempo

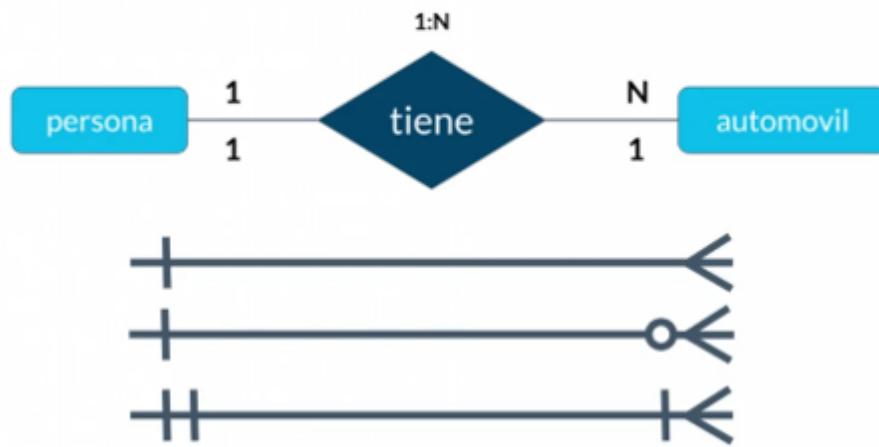
Cardinalidad: 0 a 1



Cardinalidad: 1 a N (1 a muchos)

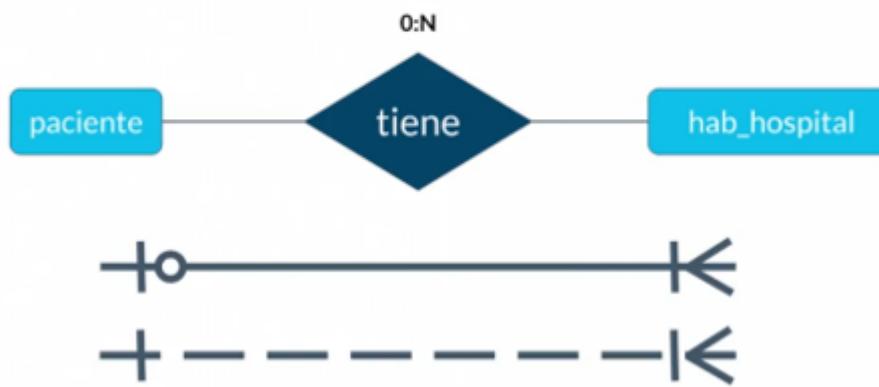
Un automóvil tiene N automóviles, pero 1 automóvil solo tiene 1 dueño.

Cardinalidad: 1 a N



Cardinalidad: 0 a N

Cardinalidad: 0 a N

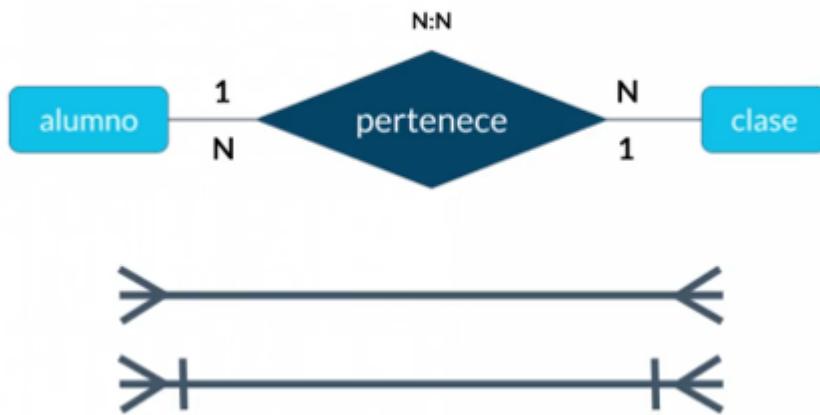


Clase 7 Multiples muchos

Este tipo de cardinalidad es muy interesante por ello su clase aparte, se vera cuando veamos los campos clave.

Cardinalidad: N a N

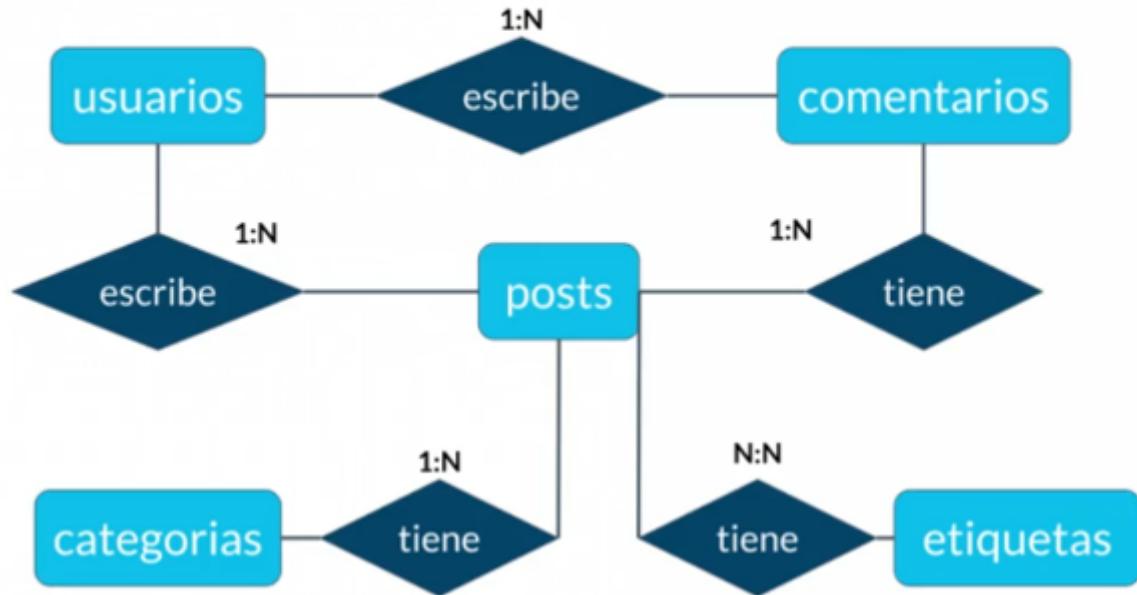
Cardinalidad: N a N



Clase 8 Diagrama ER

Un diagrama es como un mapa y nos ayuda a entender cuáles son las entidades con las que vamos a trabajar, cuáles son sus relaciones y qué papel van a jugar en las aplicaciones de la base de datos.

Diagrama ER: Platziblog



Clase 9 Diagrama Físico: tipos de datos y constraints

Para llevar a la práctica un diagrama debemos ir más allá y darle detalle con parámetros como:

Tipos de dato

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

Nota:

Char(8) reserva 8 espacios en memoria de forma fija, Varchar(8) hace lo mismo pero crece (1,2,3...8) de manera dinámica conforme los requieres.

Constraints (Restricciones)

- **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, tiene la desventaja de que tiene que reindexar los registros cada vez, lo que vuelve muy lenta la operación de la bd.

Clase 10 Diagrama Físico: normalizacion

La normalización como su nombre lo indica nos ayuda a dejar todo de una forma normal. Esto obedece a las 12 reglas de Codd y nos permiten separar componentes en la base de datos:

Primera forma normal (1FN): Atributos atómicos (Sin campos repetidos) **Segunda forma normal (2FN):** Cumple 1FN y cada campo de la tabla debe depender de una clave única. **Tercera forma normal (3FN):** Cumple 1FN y 2FN y los campos que NO son clave, NO deben tener dependencias. **Cuarta forma normal (4FN):** Cumple 1FN, 2FN, 3FN y los campos multivaluados se identifican por una clave única.

Sin normalizar

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

Primera forma normal (1FN): Atributos atómicos (Sin campos repetidos)

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

Segunda forma normal (2FN): Cumple 1FN y cada campo de la tabla debe depender de una clave única.

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

Tercera forma normal (3FN): Cumple 1FN y 2FN y los campos que NO son clave, NO deben tener dependencias.

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

Cuarta forma normal (4FN): Cumple 1FN, 2FN, 3FN y los campos multivaluados se identifican por una clave única.

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

Clase 11 Diagrama Físico: normalizando Platziblog

Diagram Entidad Relacion

Diagrama ER: Platziblog

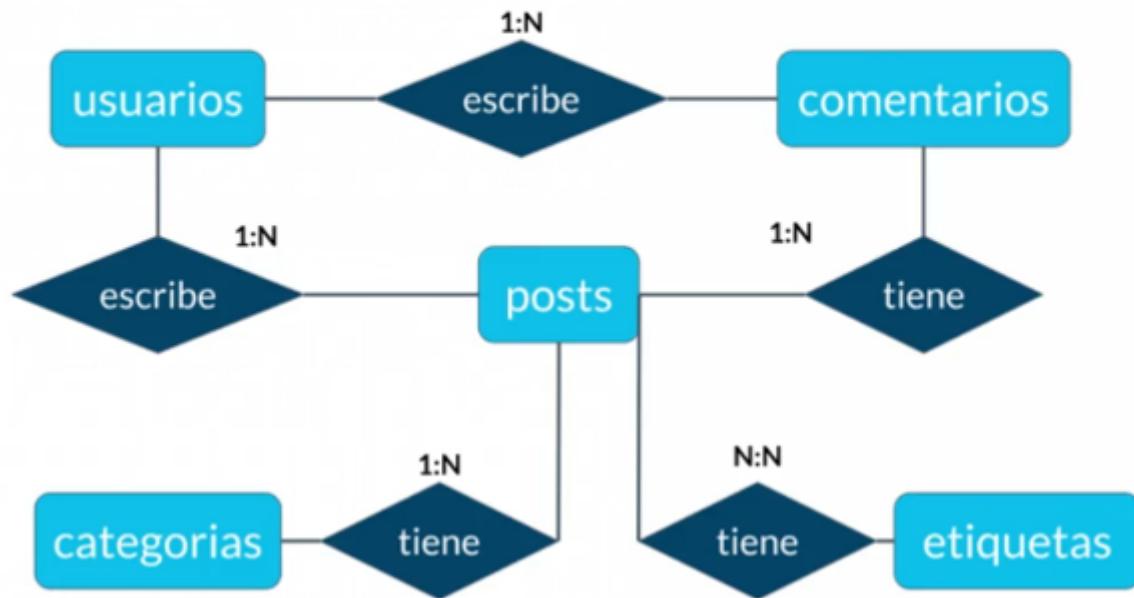


Diagrama fisico Paso 1

Generar los campos básicos de cada entidad

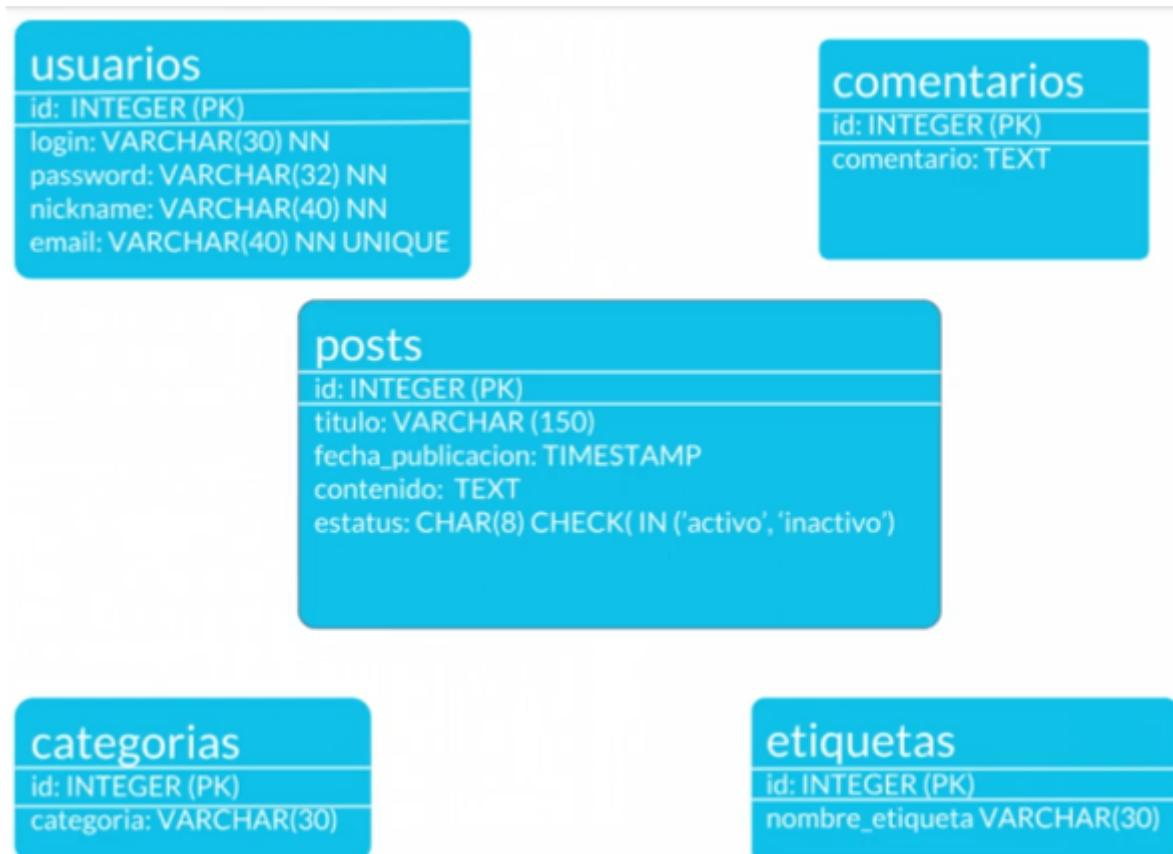


Diagrama fisico Paso 2

Genera las relaciones entre cada entidad y asigna PK y FK.

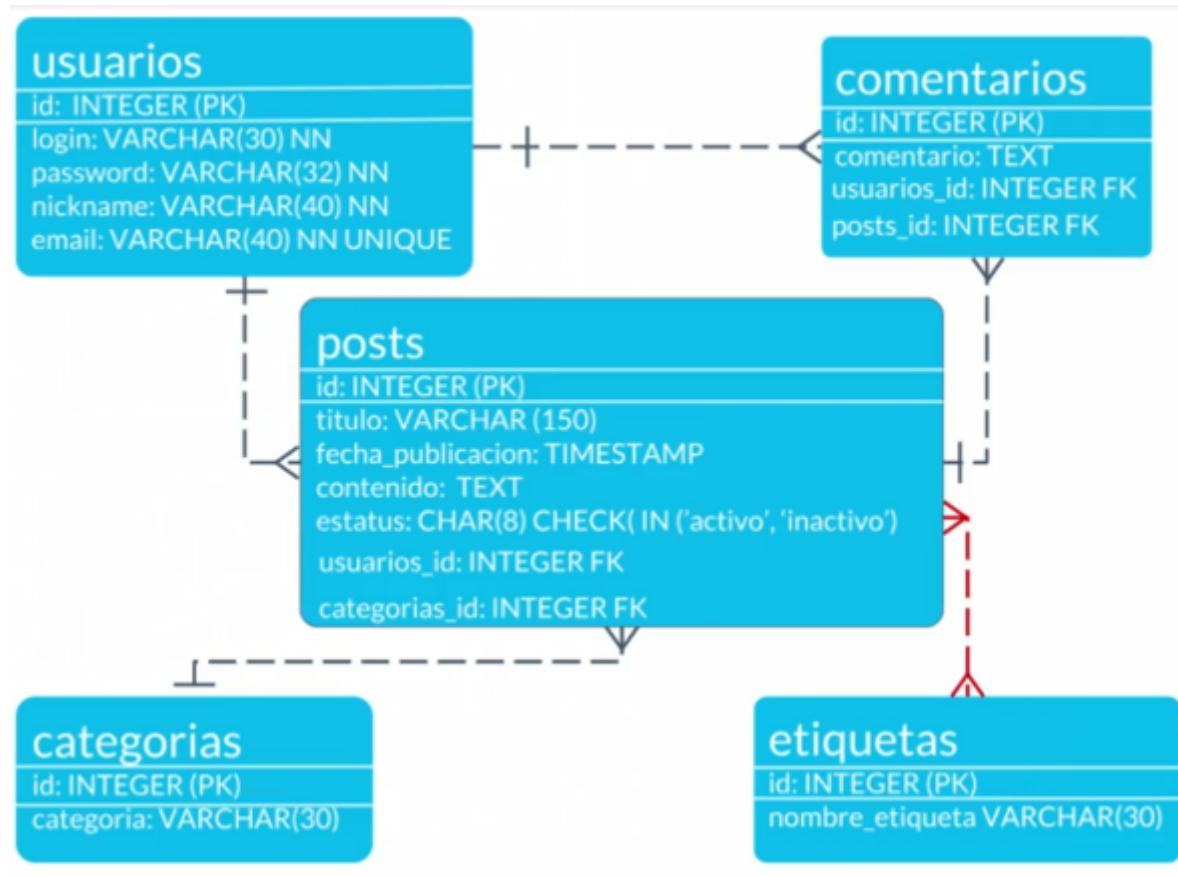
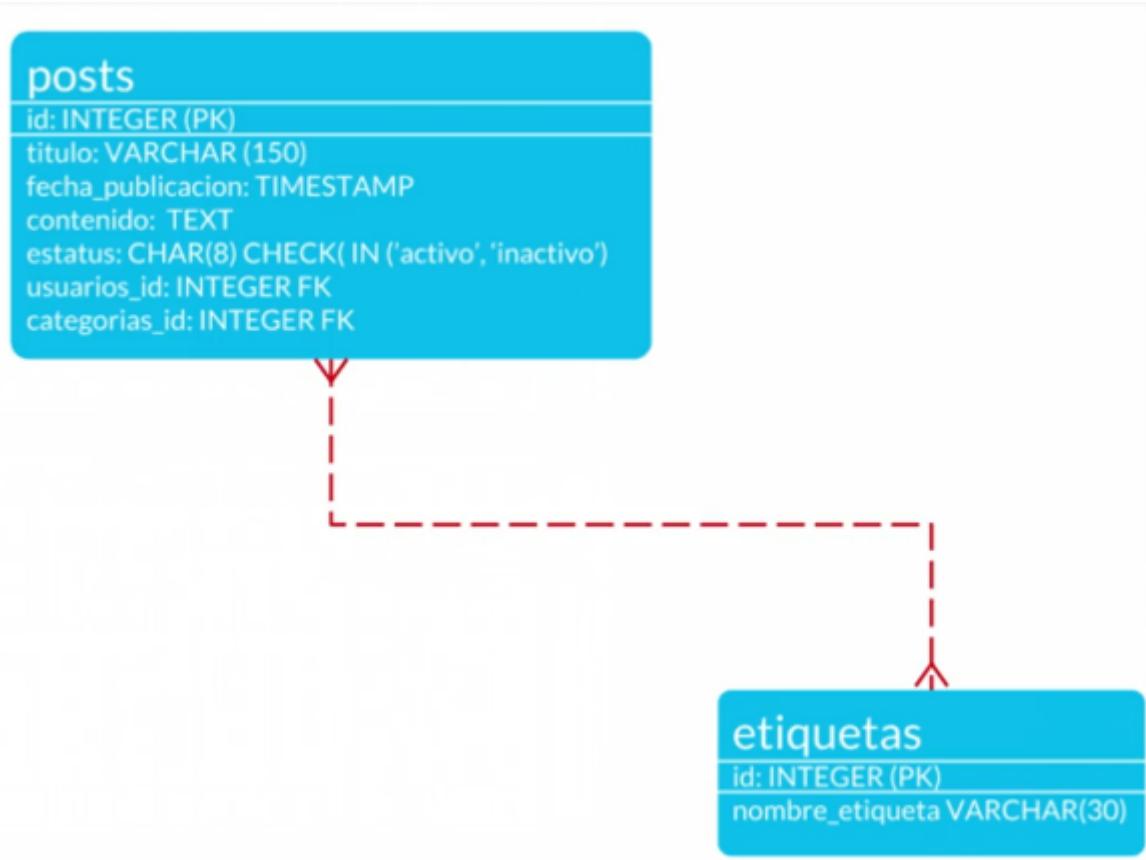
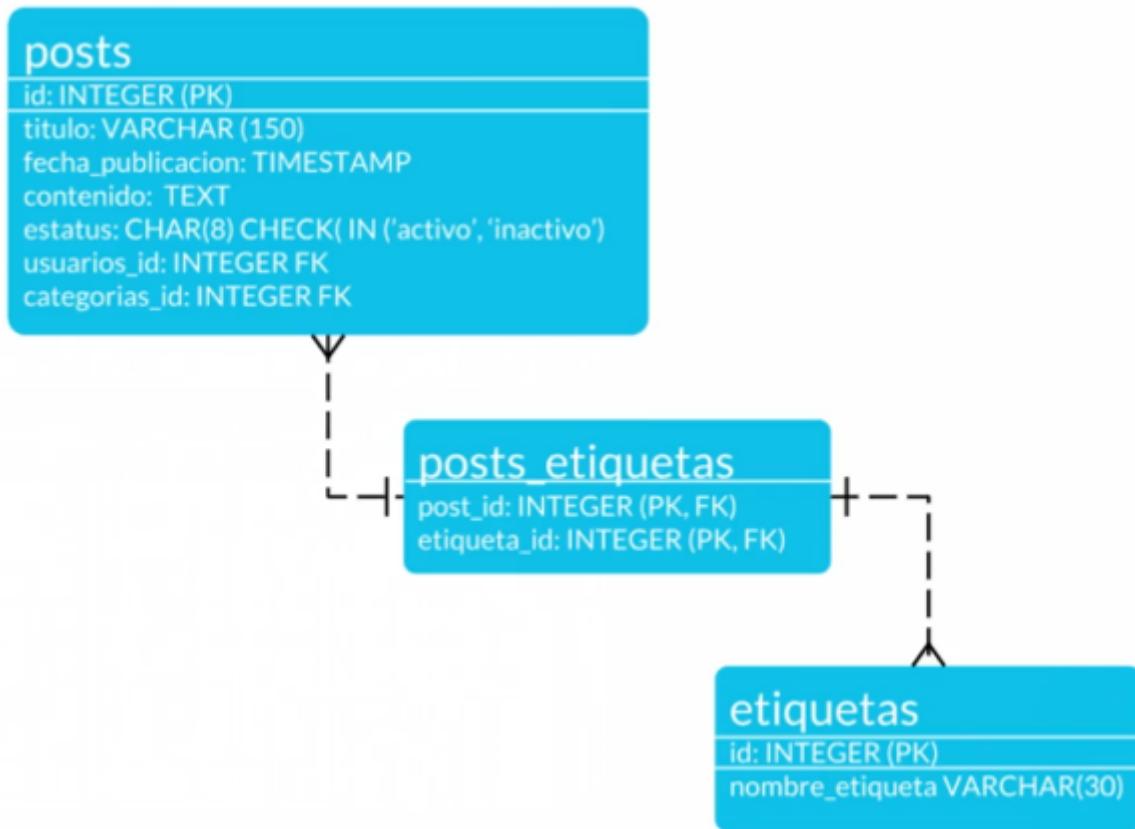


Diagrama fisico Paso 3



Para manejar las relaciones muchos a muchos, es necesario generar una tabla intermedia de esta manera.



Observa que post_id y etiqueta_id usan llaves compuestas para hacer combinaciones únicas, o podemos generar también la forma anterior con un id

Clase 12 Formas normales en DB relacionales

La normalización en las bases de datos relacionales es uno de esos temas que, por un lado es sumamente importante y por el otro suena algo esotérico. Vamos a tratar de entender las formas normales (FN) de una manera simple para que puedas aplicarlas en tus proyectos profesionales.

Primera Forma Normal (1FN)

Esta FN nos ayuda a eliminar los valores repetidos y no atómicos dentro de una base de datos.

Formalmente, una tabla está en primera forma normal si:

- Todos los atributos son atómicos. Un atributo es atómico si los elementos del dominio son simples e indivisibles.
- No debe existir variación en el número de columnas.
- Los campos no clave deben identificarse por la clave (dependencia funcional).
- Debe existir una independencia del orden tanto de las filas como de las columnas; es decir, si los datos cambian de orden no deben cambiar sus significados.

Se traduce básicamente a que si tenemos campos compuestos como por ejemplo "nombre_completo" que en realidad contiene varios datos distintos, en este caso podría ser "nombre", "apellido_paterno", "apellido_materno", etc.

Se traduce básicamente a que si tenemos campos compuestos como por ejemplo "nombre_completo" que en realidad contiene varios datos distintos, en este caso podría ser "nombre", "apellido_paterno",

"apellido_materno", etc.

Los campos deben ser tales que si reordenamos los registros o reordenamos las columnas, cada dato no pierda el significado.

Segunda Forma Normal (2FN)

Esta FN nos ayuda a diferenciar los datos en diversas entidades.

Formalmente, una tabla está en segunda forma normal si:

- Está en 1FN
- Si los atributos que no forman parte de ninguna clave dependen de forma completa de la clave principal. Es decir, que no existen dependencias parciales.
- Todos los atributos que no son clave principal deben depender únicamente de la clave principal.

Lo anterior quiere decir que sí tenemos datos que pertenecen a diversas entidades, cada entidad debe tener un campo clave separado. Por ejemplo:

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

En la tabla anterior tenemos por lo menos dos entidades que debemos separar para que cada uno dependa de manera única de su campo llave o ID. En este caso las entidades son alumnos por un lado y materias por el otro, ya que una materia. En el ejemplo anterior, quedaría de la siguiente manera:

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

Tercera Forma Normal (3FN)

Esta FN nos ayuda a separar conceptualmente las entidades que no son dependientes.

Formalmente, una tabla está en tercera forma normal si:

Se encuentra en 2FN No existe ninguna dependencia funcional transitiva en los atributos que no son clave

Esta FN se traduce en que aquellos datos que no pertenecen a la entidad deben tener una independencia de las demás y debe tener un campo clave propio. Continuando con el ejemplo anterior, al aplicar la 3FN separamos la tabla alumnos ya que contiene datos de los cursos en ella quedando de la siguiente manera.

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

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

Cuarta Forma Normal (4FN)

Esta FN nos **trata de atomizar los datos multivaluados** de manera que no tengamos datos repetidos entre rows.

Formalmente, una tabla está en cuarta forma normal si:

- Se encuentra en 3FN
- Los campos multivaluados se identifican por una clave única

Esta FN trata de eliminar registros duplicados en una entidad, es decir que cada registro tenga un contenido único y de necesitar repetir la data en los resultados se realiza a través de claves foráneas.

Aplicado al ejemplo anterior la tabla materia se independiza y se relaciona con el alumno a través de una tabla transitiva o pivote, de tal manera que si cambiamos el nombre de la materia solamente hay que cambiarla una vez y se propagara a cualquier referencia que haya de ella.

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

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

De esta manera, aunque parezca que la información se multiplicó, en realidad la descompusimos o normalizamos de manera que a un sistema le sea fácil de reconocer y mantener la consistencia de los datos.

Algunos autores precisan una 5FN que hace referencia a que después de realizar esta normalización a través de uniones (JOIN) permita regresar a la data original de la cual partió.

Modulo 3 RDBMS (MySQL) o cómo hacer lo anterior de manera práctica

Clase 13 RDB Qué

RDBMS significa Relational Database Management System o sistema manejador de bases de datos relacionales. Es un programa que se encarga de seguir las reglas de Codd y se puede utilizar de manera programática.

Clase 14 Instalación local de un RDBMS (Windows)

Hay dos maneras de acceder a manejadores de bases de datos:

- Instalar en máquina local un administrador de bases relacional. -Tener ambientes de desarrollo especiales o servicios cloud.

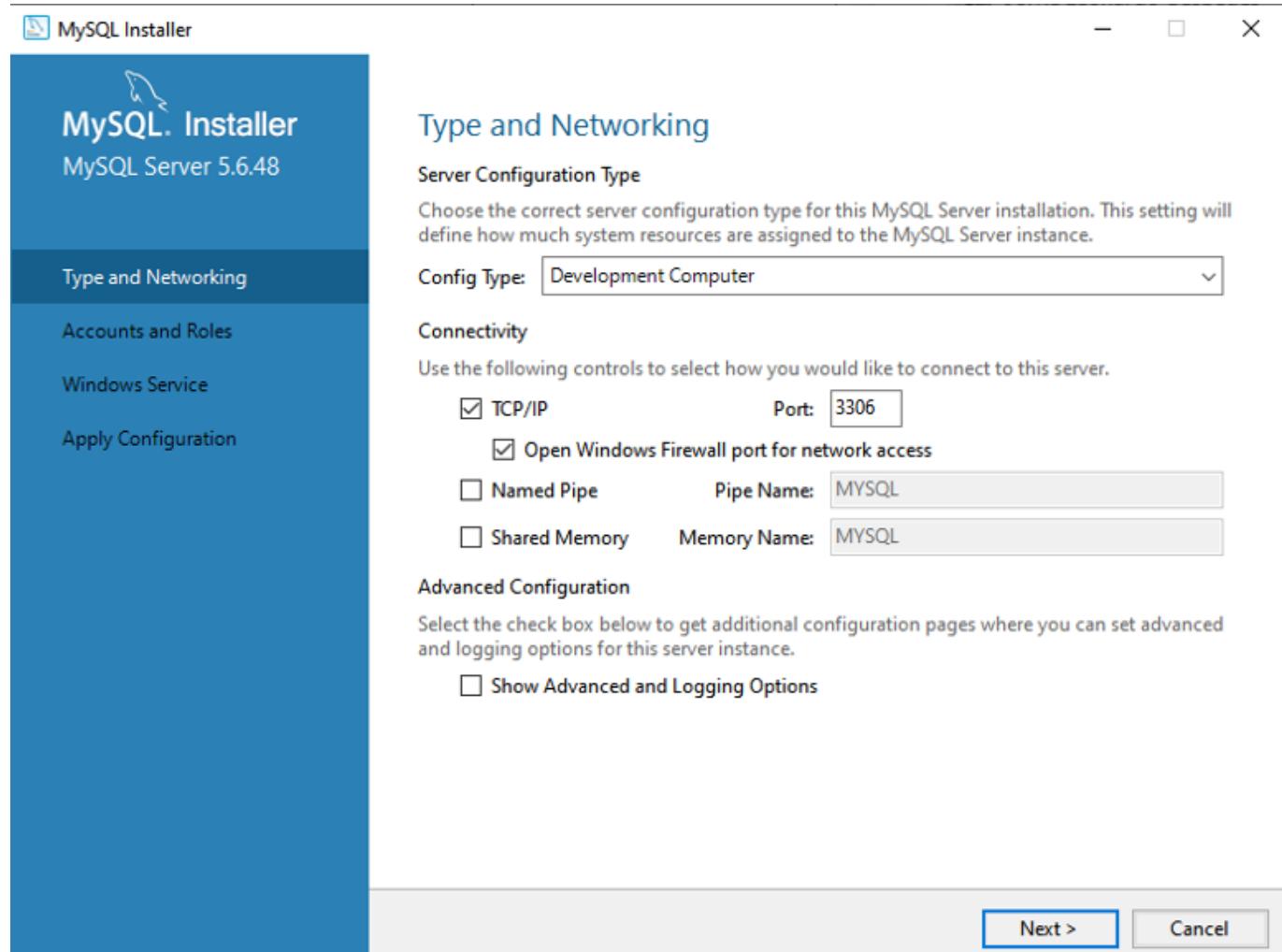
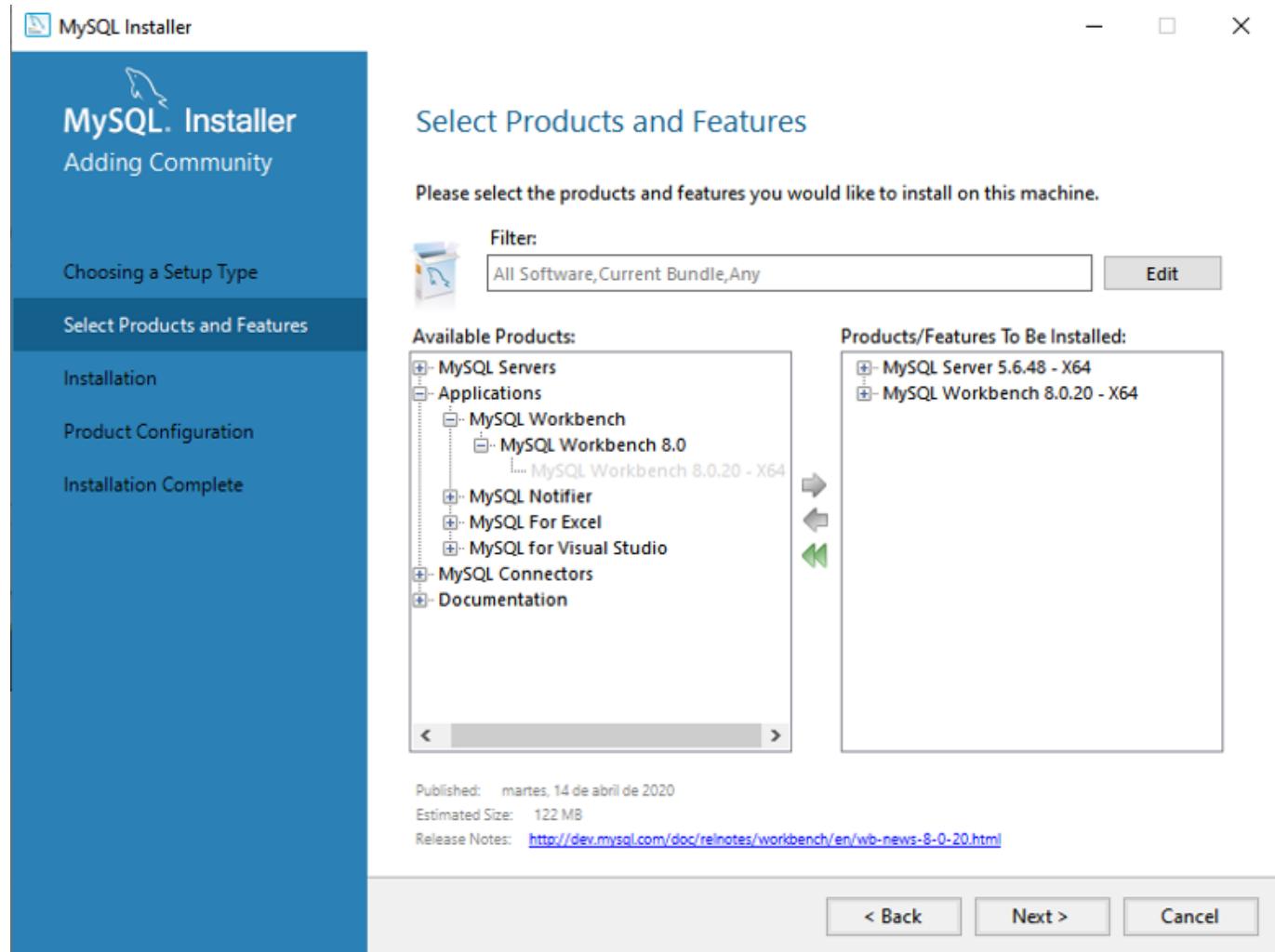
En este curso usaremos MySQL porque tiene un impacto histórico siendo muy utilizado y además es software libre y gratuito. La versión 5.6.43 es compatible con la mayoría de aplicaciones y frameworks.

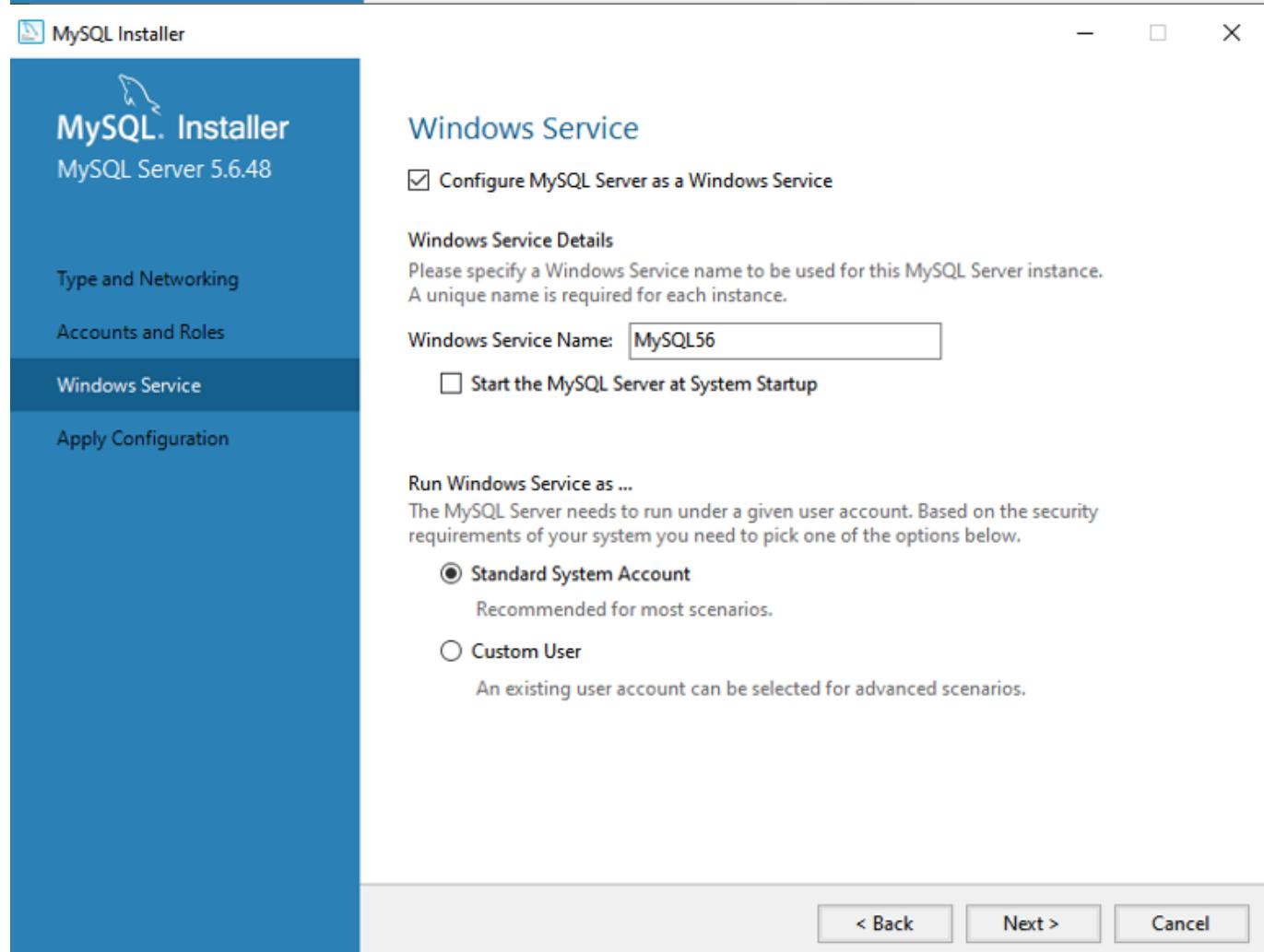
- Root es el usuario principal que tendrá todos los permisos y por lo tanto en ambientes de producción hay que tener mucho cuidado al configurarlo.

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

Procedimiento:

- Descargamos e instalamos como es usual en windows
- En el instalador seleccionamos la opción custom
- Instalamos MySQL Server la version 64bts
- Instalamos MySQL Workbench (ignoramos lo demás)





Clase 15 Instalación local de un RDBMS (Mac)

La instalación es similar al todos los instaladores en mac, descarga el archivo .dmg

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

Para macOS debes descargar workbench aparte.

<https://dev.mysql.com/downloads/mysql/5.7.html>

Clase 16 Instalación local de un RDBMS (Ubuntu)

Visita la dirección de descarga de la versión de comunidad de MySql

<https://dev.mysql.com/downloads/mysql/5.7.html#downloads>

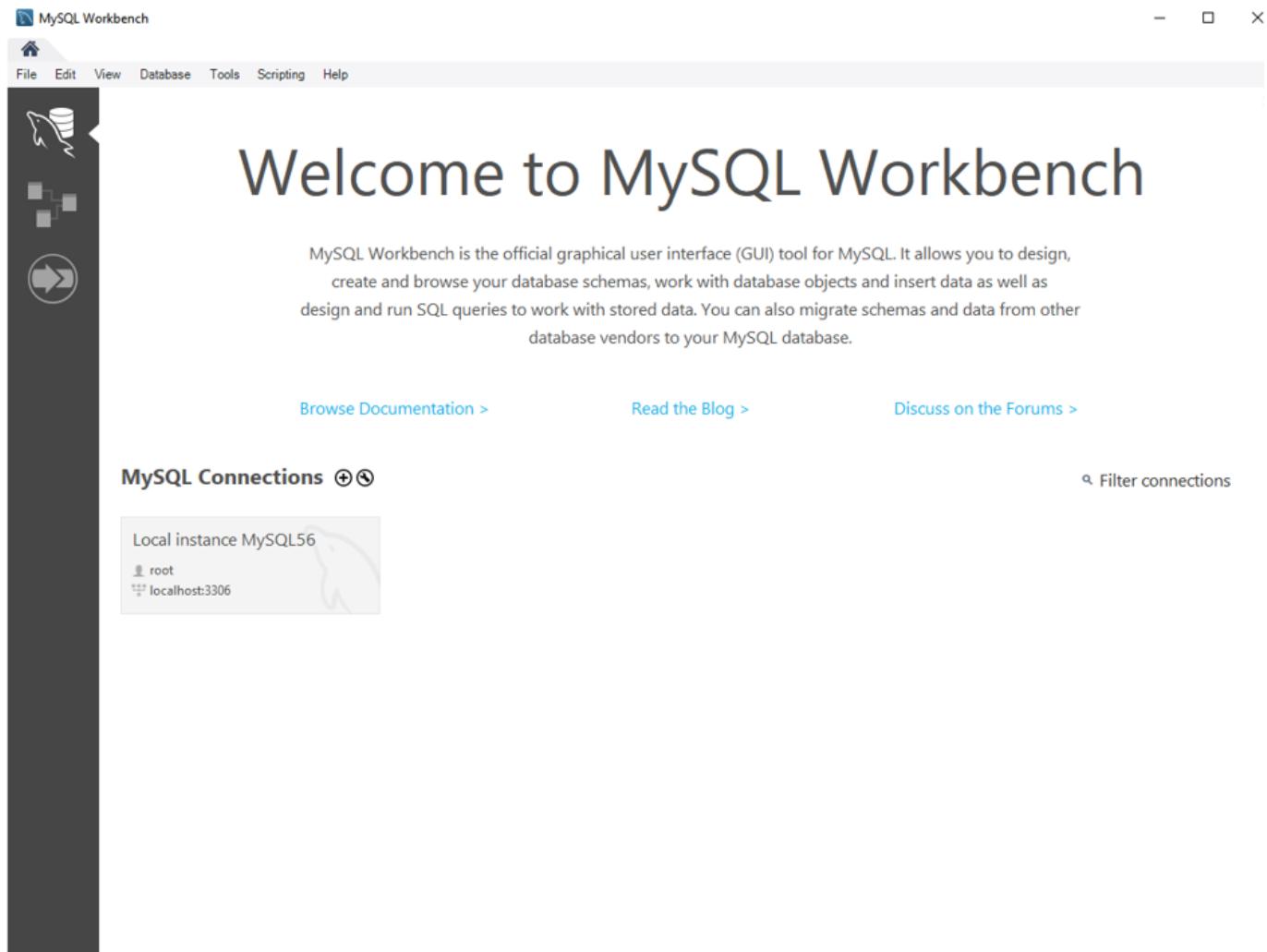
Dirígete a la sección de selección de descargas y selecciona tu distribución de Linux. En nuestro caso Ubuntu y selecciona posteriormente la versión que estás utilizando actualmente, en nuestro caso 18.04 de 64 bits.

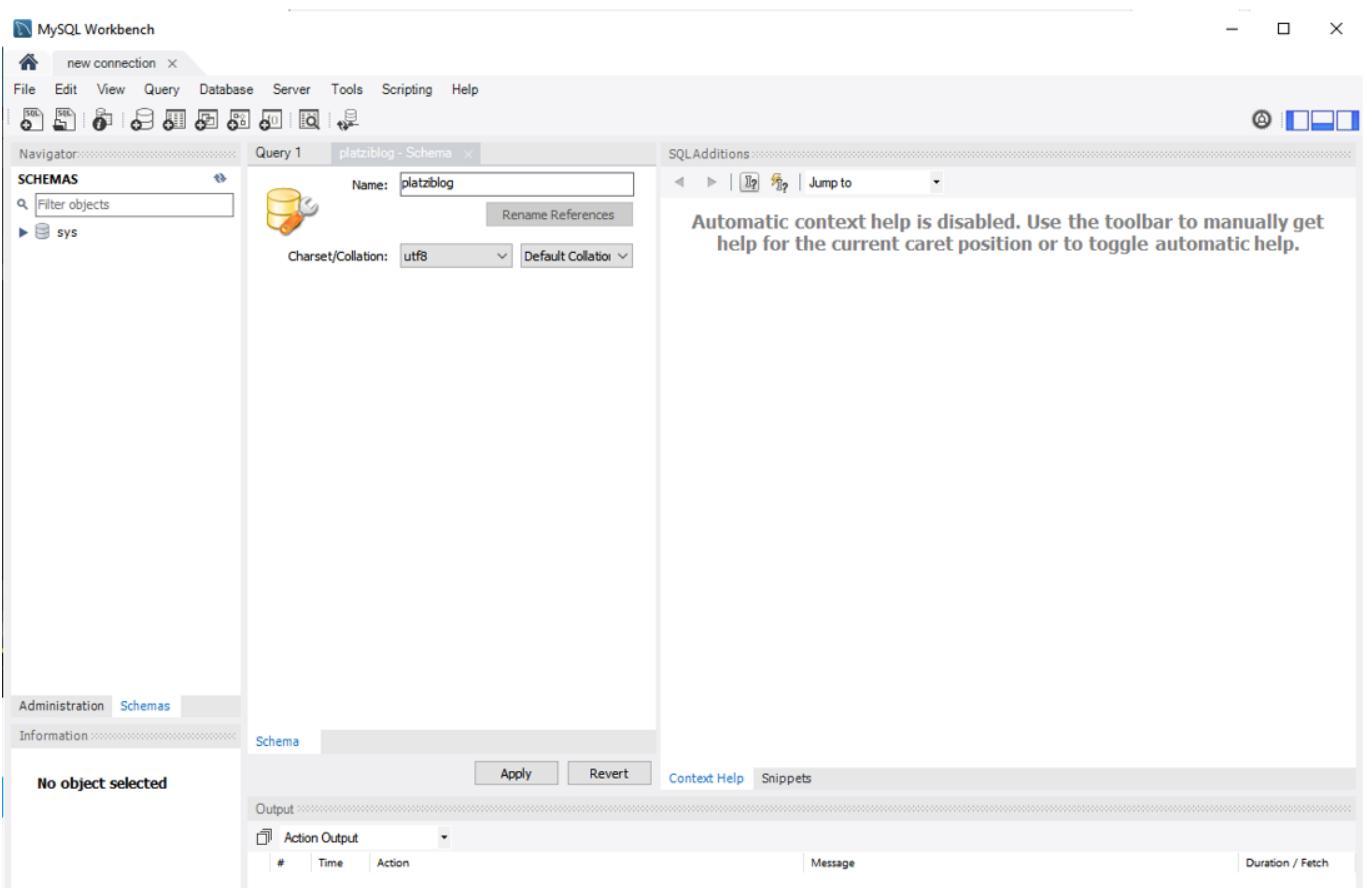
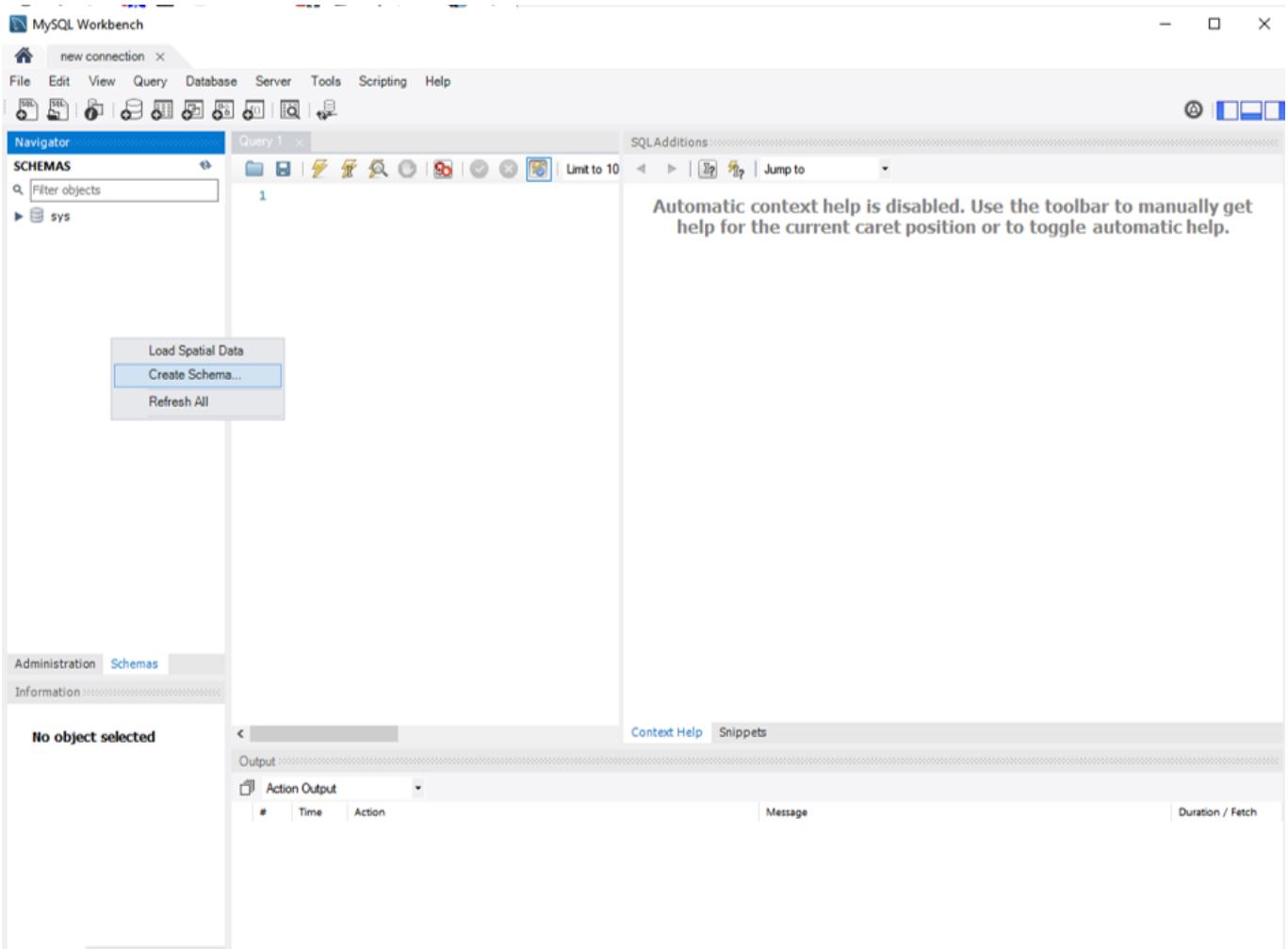
Los pasos son similares a los otros sistemas, descarga el paquete .deb, también puedes instalar desde la consola

```
sudo apt-get install mysql-server
```

Clase 17 Clientes Graficos

Observamos el preview y pasos para crear un schema en Mysql workbench





Hoy en día muchas empresas ya no tienen instalados en sus servidores los RDBMS sino que los contratan a otras personas. Estos servicios administrados cloud te permiten concentrarte en la base de datos y no en su administración y actualización.

Introducción a google cloud, toma el curso para configurarlo.

Modulo 4 SQL hasta en la sopa

Clase 19 Historia de SQL

SQL significa Structured Query Language y tiene una estructura clara y fija. Su objetivo es hacer un solo lenguaje para consultar cualquier manejador de bases de datos volviéndose un gran estándar.

Ahora existe el **NOSQL** o Not Only Structured Query Language que significa que no sólo se utiliza SQL en las bases de datos no relacionales.

Clase 20 DDL create

SQL tiene dos grandes sublenguajes:

DDL o Data Definition Language que nos ayuda a crear la estructura de una base de datos. Existen 3 grandes comandos:

- **Create**: Nos ayuda a crear bases de datos, tablas, vistas, índices, etc.
- **Alter**: Ayuda a alterar o modificar entidades.
- **Drop**: Nos ayuda a borrar. Hay que tener cuidado al utilizarlo.

3 objetos que manipularemos con el lenguaje DDL:

- Database o bases de datos
- Table o tablas. Son la traducción a SQL de las entidades
- View o vistas: Se ofrece la proyección de los datos de la base de datos de forma entendible.

Create

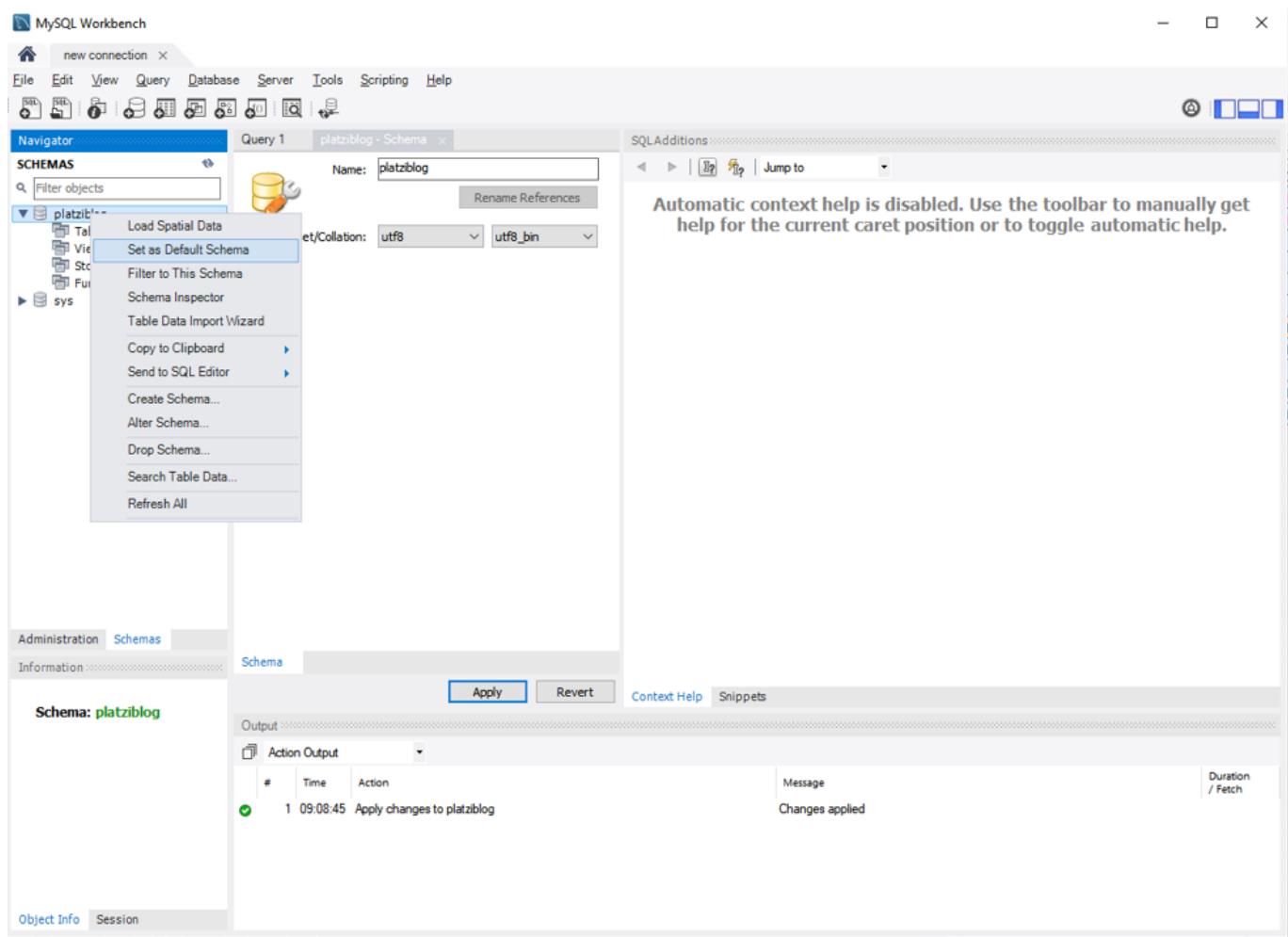


Create Database

```
CREATE DATABASE test_db;

USE DATABASE test_db;
```

Seleccionamos el schema como default, de forma gráfica aunque con la terminal seria el use database.



El comando Create Table

Create Table

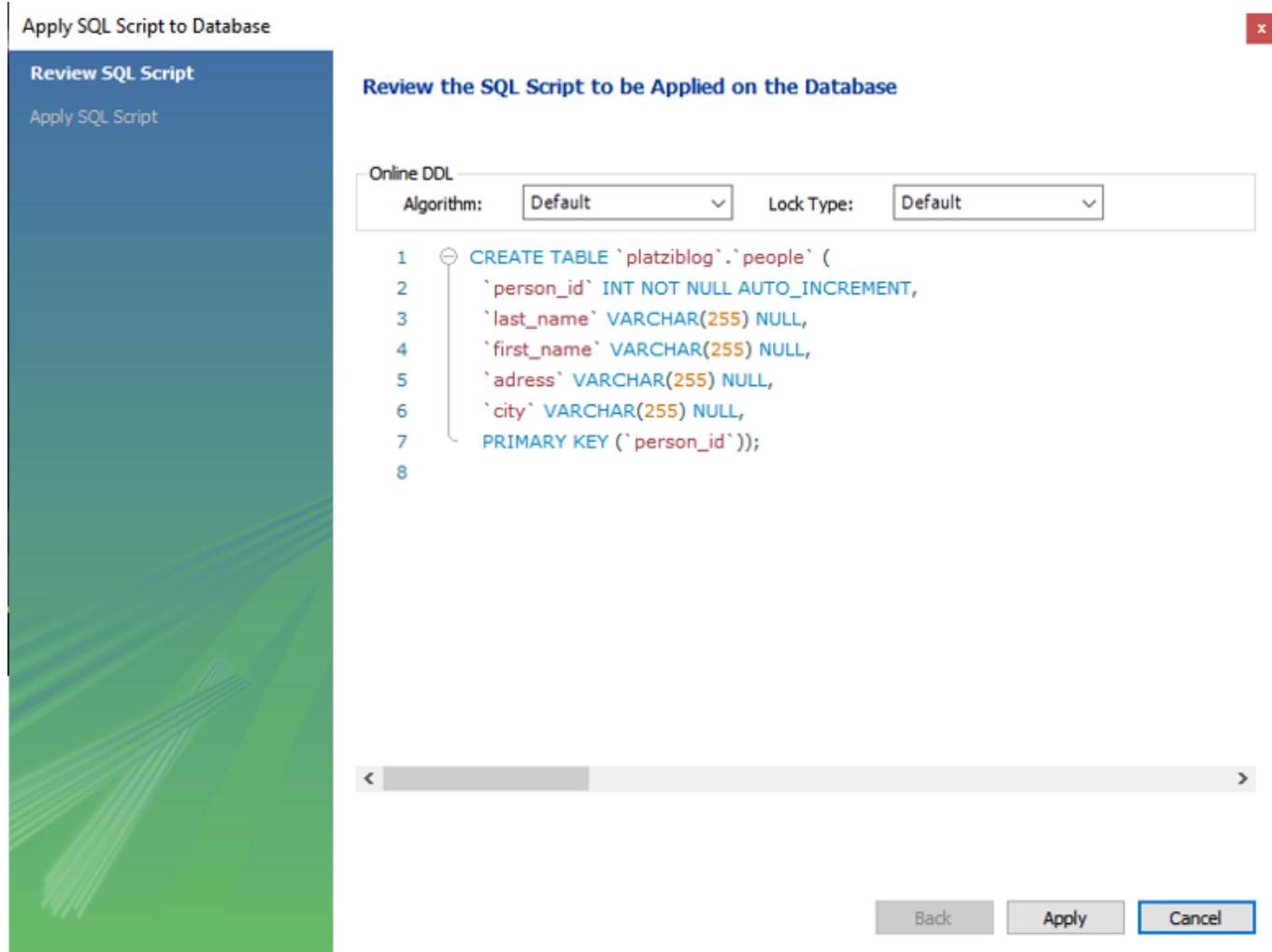
```
CREATE TABLE people (
    person_id int,
    last_name varchar(255),
    first_name varchar(255),
    address varchar(255),
    city varchar(255)
);
```

El creamos con workbench la tabla anterior, definimos cada uno de los campos de manera sencilla posterior damos apply

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

- Navigator:** Shows the 'platziblog' schema selected under 'SCHEMAS'.
- Table Definition:**
 - Table Name:** people
 - Schema:** platziblog
 - Charset/Collation:** Default Charset
 - Engine:** InnoDB
 - Comments:** (empty)
- Columns:**

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
person_id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>					
last_name	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
first_name	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
address	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
city	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
- Storage Options:**
 - Storage:** Virtual, Primary Key, Binary, Auto Increment (checkboxes)
 - Not Null, Unique, Unsigned, Zero Fill, Generated (checkboxes)**
- Output:**
 - Action Output: # Time Action
 - Message: Changes applied
 - Duration / Fetch



MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

platziblog

Tables

people

Query 1

```
1 • SELECT * FROM platziblog.people;
```

Result Grid

person_id	last_name	first_name	adress	city
HULL	HULL	HULL	HULL	HULL

Administration Schemas

Information

Table: people

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	09:08:45	Apply changes to platziblog	Changes applied	
2	21:17:18	Apply changes to people	Changes applied	
3	21:44:53	use table people	Error Code: 1064. You have an error in your SQL syntax; check the manual...	0.000 sec
4	21:44:58	use people	Error Code: 1049. Unknown database 'people'	0.000 sec
5	21:45:09	tables	Error Code: 1064. You have an error in your SQL syntax; check the manual...	0.000 sec
6	21:45:15	SELECT * FROM platziblog.people LIMIT 0, 1000	0 row(s) returned	0.015 sec / 0.000 sec

Object Info Session

Clase 21 CREATE VIEW y DDL ALTER

Create view

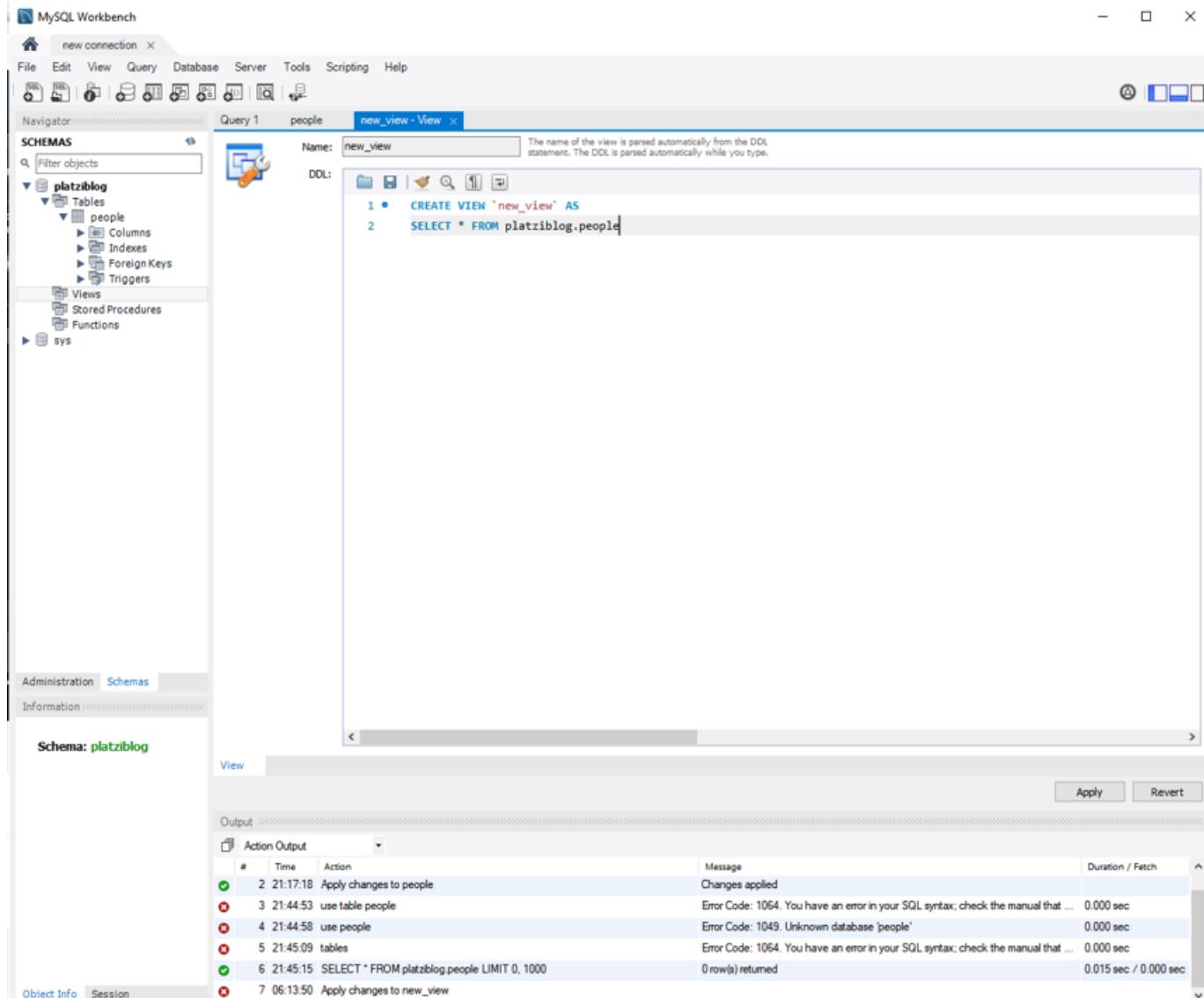
Create View

```
CREATE VIEW v_brasil_customers AS
    SELECT customer_name,
    contact_name
    FROM customers
    WHERE country = "Brasil";
```

Para iniciar el ejercicio insertamos datos en la BD, copiamos la sentencia `SELECT * FROM platziblog.people;`, nos movemos a views y damos click derecho y create view.

Views ya cuenta con la parte inicial de la sentencia de create view, pegamos debajo el select y damos apply

```
CREATE VIEW 'new_view' AS
SELECT * FROM platziblog.people;
```



Apply SQL Script to Database X

Review SQL Script X

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

Algorithm: Default Lock Type: Default

```
1 USE `platziblog`;
2 CREATE OR REPLACE VIEW `new_view` AS
3 SELECT * FROM platziblog.people;
4
```

< >

Back Apply Cancel

The screenshot shows a MySQL Workbench dialog box. The title bar says 'Apply SQL Script to Database'. On the left, there's a sidebar with 'Review SQL Script' and 'Apply SQL Script' buttons. The main area is titled 'Review the SQL Script to be Applied on the Database'. It shows an 'Online DDL' section with dropdown menus for 'Algorithm' (set to 'Default') and 'Lock Type' (set to 'Default'). Below that is a code editor containing the following SQL script:

```
1 USE `platziblog`;
2 CREATE OR REPLACE VIEW `new_view` AS
3 SELECT * FROM platziblog.people;
4
```

At the bottom of the dialog are 'Back', 'Apply', and 'Cancel' buttons.

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help' are visible. The 'Query 1' tab is selected, showing a 'new_view - View' window. The 'Name:' field contains 'new_view'. Below it, the DDL code for creating the view is displayed:

```
1 • CREATE
2     ALGORITHM = UNDEFINED
3     DEFINER = 'root'@'localhost'
4     SQL SECURITY DEFINER
5     VIEW `new_view` AS
6         SELECT
7             `people`.`person_id` AS `person_id`,
8             `people`.`last_name` AS `last_name`,
9             `people`.`first_name` AS `first_name`,
10            `people`.`adress` AS `adress`,
11            `people`.`city` AS `city`
12        FROM
13            `people`
```

The left sidebar shows the 'Schemas' tree, with 'platziblog' selected, displaying 'Tables' like 'people', 'Columns', 'Indexes', 'Foreign Keys', 'Triggers', 'Views', 'Stored Procedures', and 'Functions'. The 'sys' schema is also listed. The bottom left shows 'Administration' and 'Schemas' tabs, with 'Information' and 'No object selected' below. The bottom right has 'View', 'Apply', and 'Revert' buttons. The 'Output' section at the bottom lists actions and their messages:

Action	Time	Message	Duration / Fetch
use table people	3 21:44:53	Error Code: 1064. You have an error in your SQL syntax; check the manual that ...	0.000 sec
use people	4 21:44:58	Error Code: 1049. Unknown database 'people'	0.000 sec
tables	5 21:45:09	Error Code: 1064. You have an error in your SQL syntax; check the manual that ...	0.000 sec
SELECT * FROM platziblog.people LIMIT 0, 1000	6 21:45:15	0 row(s) returned	0.015 sec / 0.000 sec
Apply changes to new_view	7 06:13:50		
Apply changes to new_view	8 06:16:47	Changes applied	

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'Query 1' is selected, and the query window displays the SQL command: `SELECT * FROM platziblog.platzi_people;`. The results grid shows three rows of data:

	person_id	last_name	first_name	adress	city
▶	1	Vázquez	Israel	Calle Famosa Num 1	México
▶	2	Hernandez	onica	Reforma 222	Mexico
▶	3	Alanis	Edgar	Central 1	Monterrey

The left sidebar shows the database schema: 'platziblog' contains 'Tables' (people) and 'Views' (platzi_people). The 'Information' panel at the bottom shows the definition of the 'platzi_people' view:

```

View: platzi_people

Columns:
person_id int(11)
last_name varchar(255)
first_name varchar(255)
adress varchar(255)
city varchar(255)

```

The 'Output' panel at the bottom right shows the history of changes applied to the view:

#	Time	Action	Message	Duration / Fetch
7	06:13:50	Apply changes to new_view		
8	06:16:47	Apply changes to new_view	Changes applied	
9	06:18:24	Apply changes to new_view	No changes detected	
10	06:19:06	Apply changes to new_view	No changes detected	
11	06:20:09	Analy changes to platzi_people	Changes applied	

Parece redundante la información, pero podemos estructurar las consultas junto con otras tablas, y estas vistas mantendrán la consulta sin necesidad de acer la de nuevo incrementando los datos de forma automática.

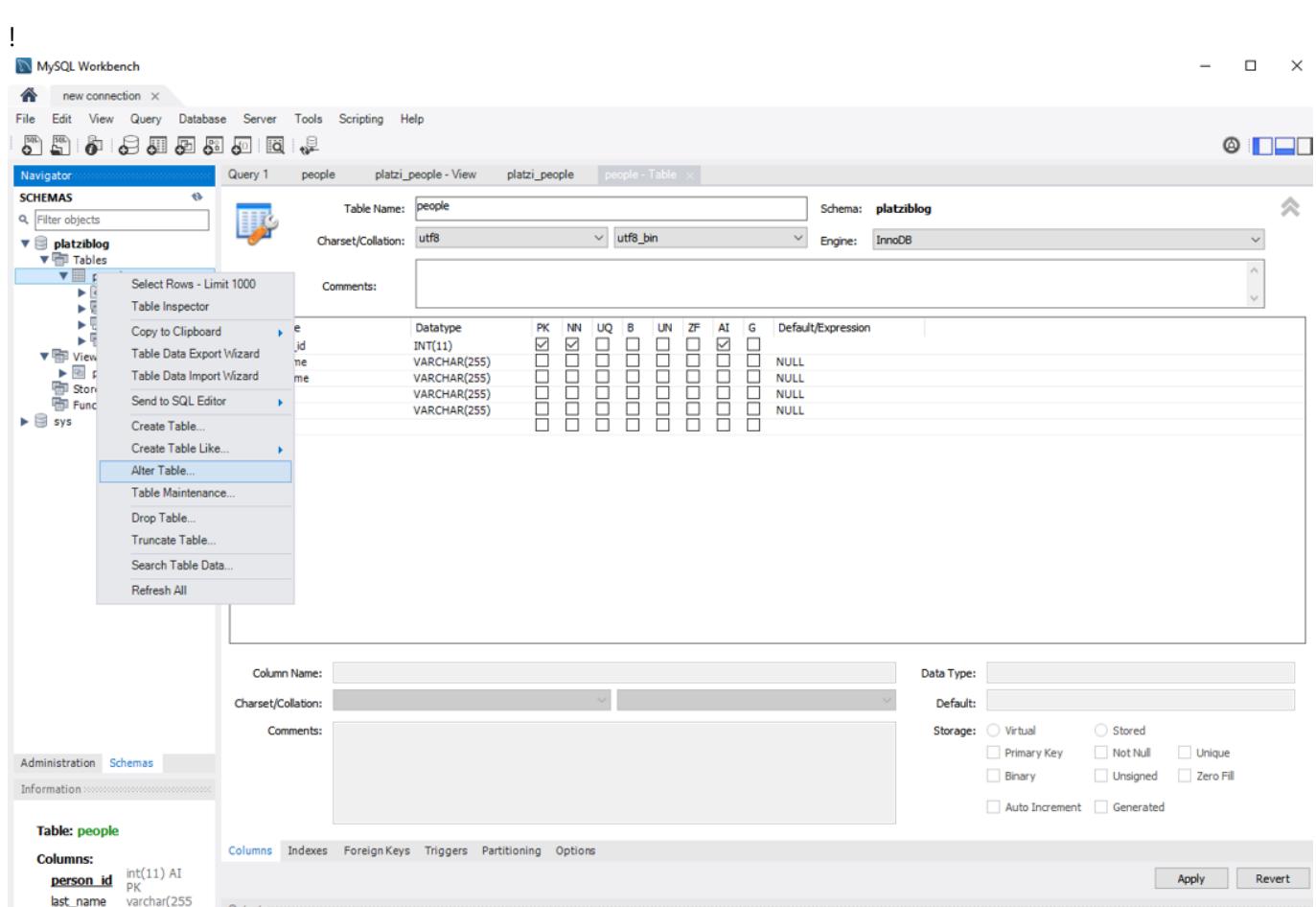
Alter Table

Alter Table

```
ALTER TABLE people  
ADD date_of_birth date;
```

```
ALTER TABLE people  
ALTER COLUMN date_of_birth year;
```

```
ALTER TABLE people  
DROP COLUMN date_of_birth;
```



MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: people platz_iblog - View platz_iblog - people - Table

Table Name: people Schema: platz_iblog
Charset/Collation: utf8 Engine: InnoDB
Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
person_id	INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL				
last_name	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
first_name	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
adress	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
city	VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
birthday	DATETIME	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Name: birthday Data Type: VARCHAR(45)
Charset/Collation: Default Charset Default Collation
Comments:
Storage: Virtual Stored
 Primary Key Not Null Unique
 Binary Unsigned Zero Fill
 Auto Increment Generated

Columns Indexes Foreign Keys Triggers Partitioning Options

Table: people

Columns:

person_id	int(11) AI PK
last_name	varchar(255)
first_name	varchar(255)
adress	varchar(255)
city	varchar(255)

Object Info Session

Output:

#	Time	Action	Message	Duration / Fetch
7	06:13:50	Apply changes to new_view	Changes applied	
8	06:16:47	Apply changes to new_view	No changes detected	
9	06:18:24	Apply changes to new_view	No changes detected	
10	06:19:06	Apply changes to new_view	Changes applied	
11	06:20:09	Apply changes to platz_iblog.platz_iblog.people	Changes applied	
12	06:20:17	SELECT * FROM platz_iblog.platz_iblog.people LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

Apply Revert

!

Apply SQL Script to Database

[Review SQL Script](#)[Apply SQL Script](#)

Review the SQL Script to be Applied on the Database

Online DDL

Algorithm:

Default

Lock Type:

Default

```
1   ALTER TABLE `platziblog`.`people`  
2     ADD COLUMN `birthday` DATETIME NULL AFTER `city`;  
3
```

[Back](#)[Apply](#)[Cancel](#)

Alter Table

```
ALTER TABLE people  
ADD date_of_birth date;
```

```
ALTER TABLE people  
ALTER COLUMN date_of_birth year;
```

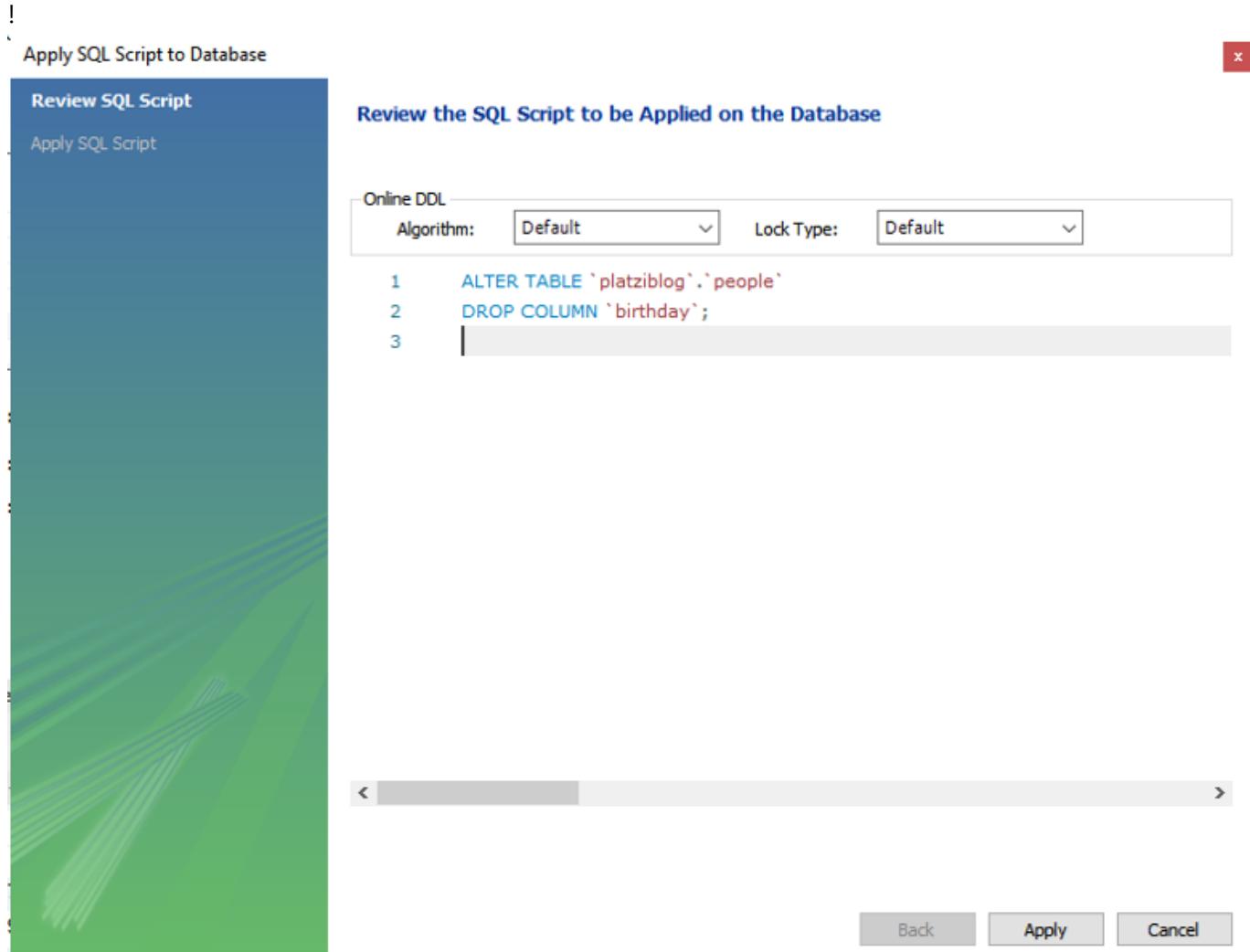
```
ALTER TABLE people  
DROP COLUMN date_of_birth;
```

!

Drop Column Borrando una columna

Click derecho y dele Selected

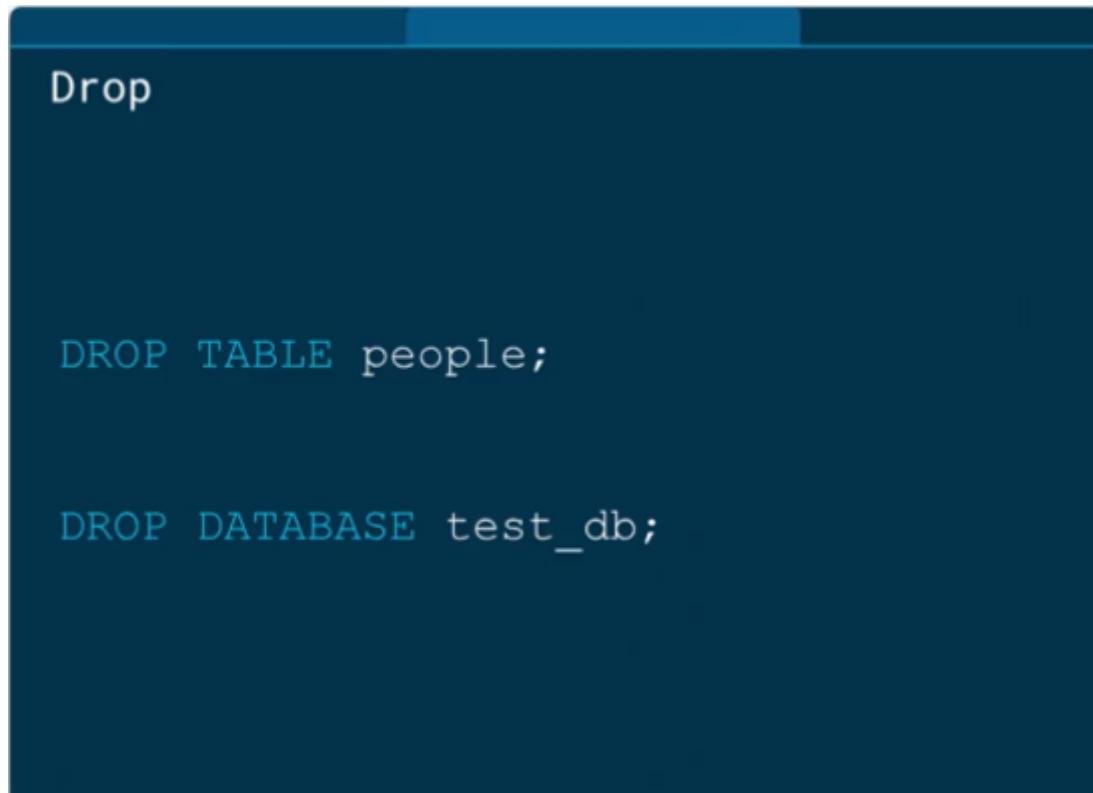
The screenshot shows the MySQL Workbench interface. On the left, the Navigator pane displays the schema 'platziblog' with its tables, views, and stored procedures. The central area shows the 'people' table definition. The 'Columns' tab is selected, showing columns: person_id (INT(11), AI PK), last_name (VARCHAR(255)), first_name (VARCHAR(255)), adress (VARCHAR(255)), city (VARCHAR(255)), and birthday (DATETIME). A context menu is open over the 'birthday' column, with the option 'Delete Selected' highlighted. The 'Table Name' is set to 'people', 'Schema' to 'platziblog', 'Charset/Collation' to 'utf8 utf8_bin', and 'Engine' to 'InnoDB'. The 'Comments' field is empty.



Clase 22 DDL drop

Está puede ser la sentencia ¡más peligrosa! (????), sobre todo cuando somos principiantes. Básicamente borra o desaparece de nuestra base de datos algún elemento.

Pasos para borrar una tabla



MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS platziblog

Tables people

Table Name: people Schema: platziblog Charset/Collation: utf8 utf8_bin Engine: InnoDB

Comments:

Datatype PK NN UQ B UN ZF AI G Default/Expression

INT(11)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL
VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL
VARCHAR(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	NULL

Select Rows - Limit 1000

Table Inspector

Copy to Clipboard

Table Data Export Wizard

Table Data Import Wizard

Send to SQL Editor

Create Table...

Create Table Like...

Alter Table...

Table Maintenance...

Drop Table...

Truncate Table...

Search Table Data...

Refresh All

Administration Schemas

Information Columns Indexes Foreign Keys Triggers Partitioning Options

Output

Action Output

#	Time	Action	Message	Duration / Fetch
12	06:20:17	SELECT * FROM platziblog.platzi_people LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
13	06:27:49	Apply changes to people	Changes applied	
14	06:38:36	Apply changes to people	No changes detected	
15	06:39:02	Apply changes to people	No changes detected	
16	06:40:23	Apply changes to people	No changes detected	
17	06:40:33	Apply changes to people	Changes applied	

Object Info Session

The screenshot shows the MySQL Workbench interface. In the center, a modal dialog titled "Drop Table" is open, asking for confirmation to permanently delete the "people" table and its data. Below the confirmation message are two buttons: "Review SQL" and "Drop Now". At the bottom right of the modal is a "Cancelar" button. The background shows the "Table Editor" for the "people" table, which has columns: person_id (PK), last_name, first_name, adress, and city. The "Columns" tab is selected. The "Information" pane at the bottom shows the table structure and some recent actions in the "Action Output" log.

This screenshot shows the MySQL Workbench interface again. A modal dialog titled "Review SQL Code to Execute" is open, displaying the SQL code: "DROP TABLE `platzi_blog`.`people`;". The "Execute" button is highlighted in blue at the bottom right of the dialog. The background shows the same "Table Editor" for the "people" table and the "Information" pane with the "Action Output" log.

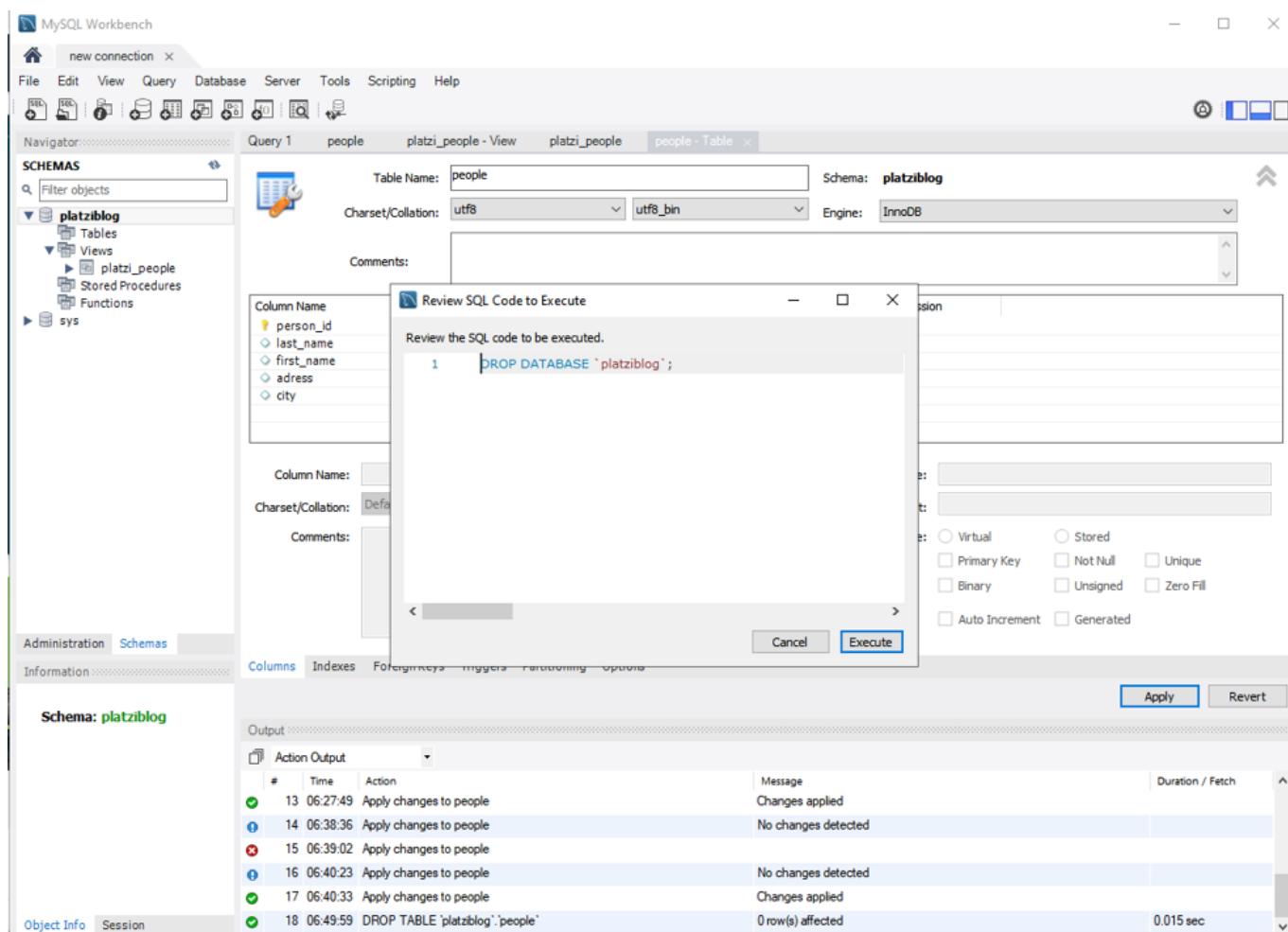
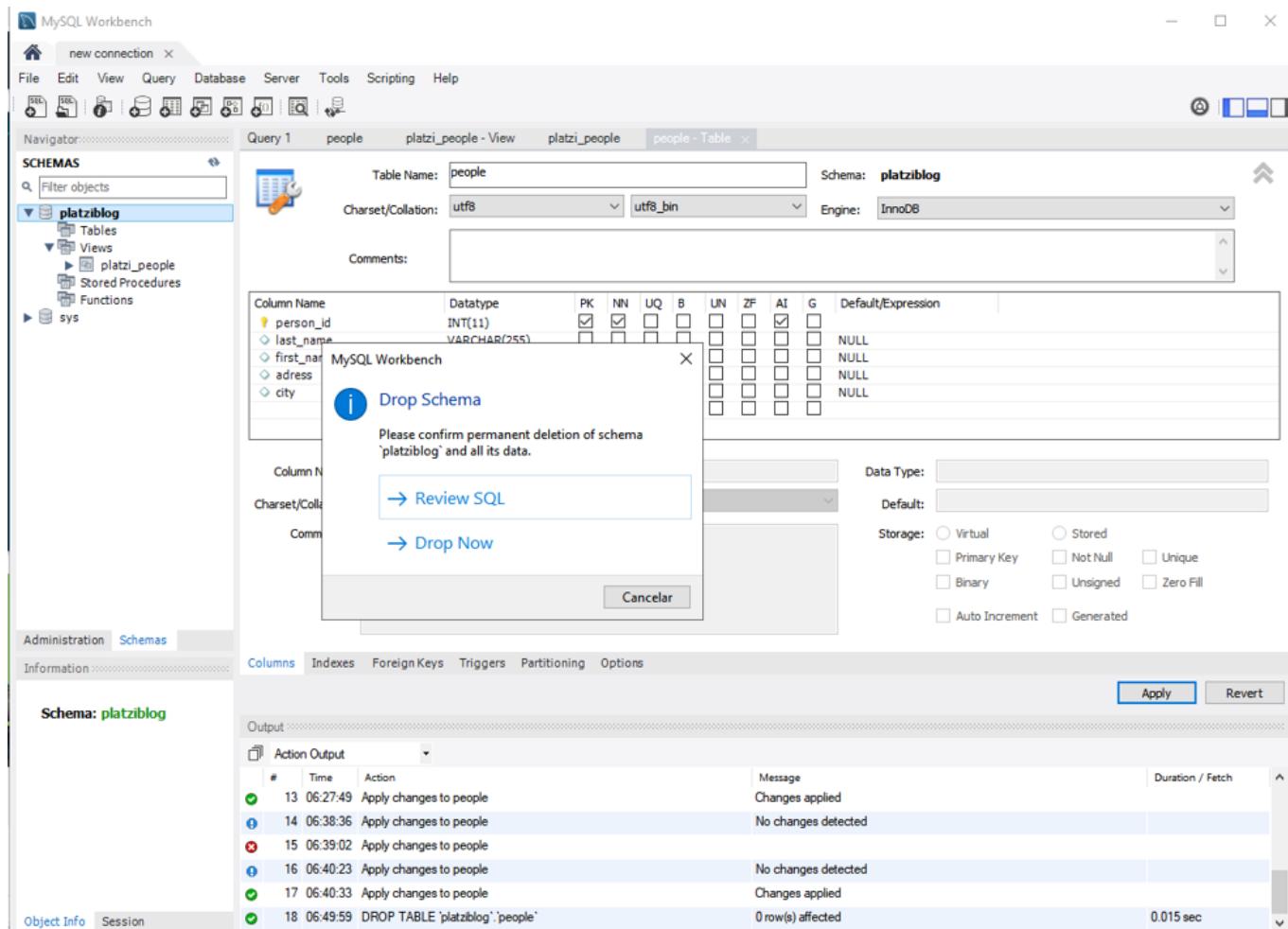
Pasos para borrar la base de datos o schema

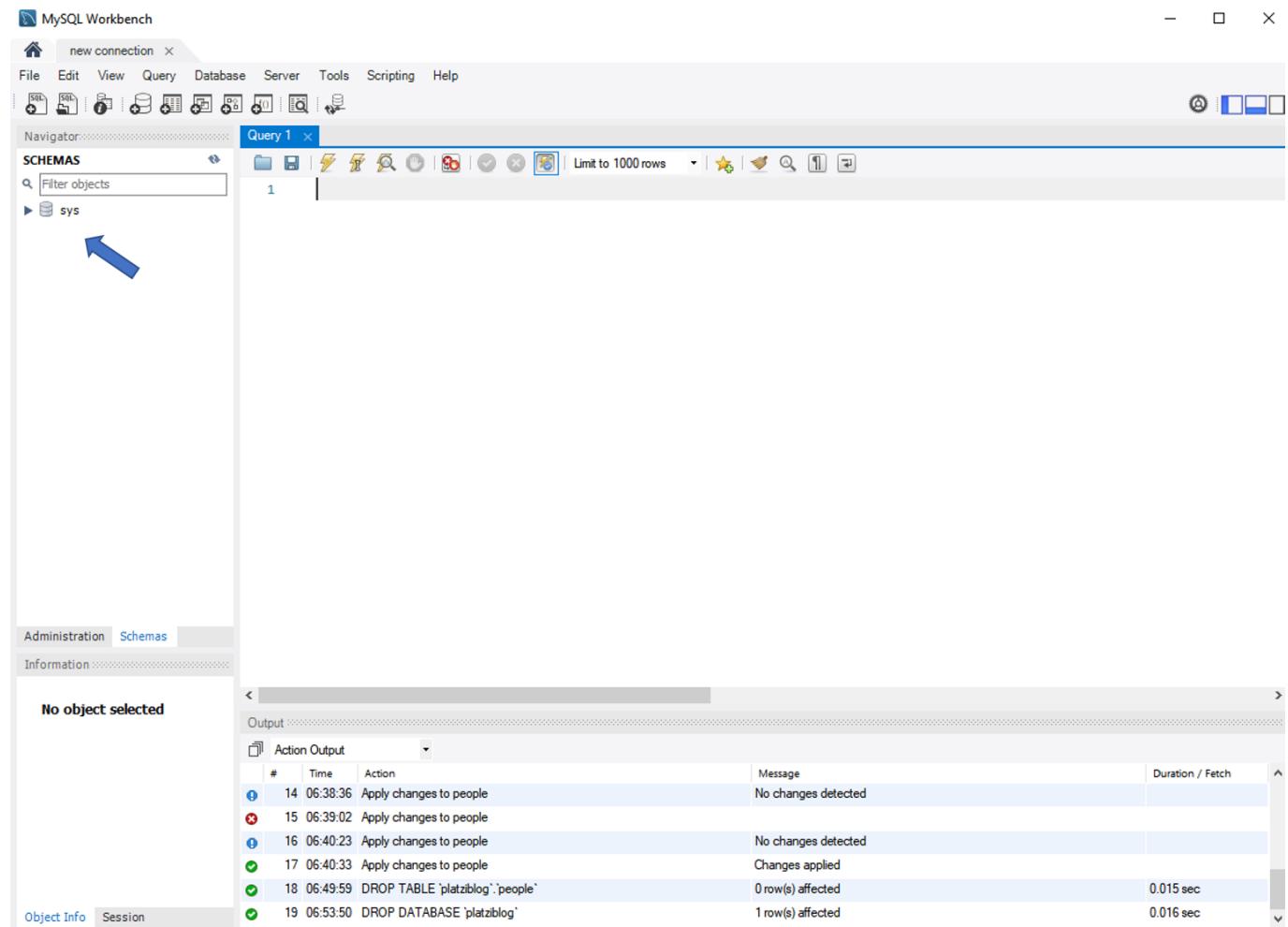
The screenshot shows the MySQL Workbench interface. In the left sidebar, under the 'Schemas' section, the 'platzi_blog' schema is selected. A context menu is open over the 'people' table, with the 'Drop Schema...' option highlighted.

The main workspace displays the 'people' table definition. The table has four columns: 'id', 'name', 'name', and 'name'. The 'id' column is defined as INT(11) with a Primary Key (PK), Not Null (NN), Unique (UQ), Binary (B), Unsigned (UN), Zero Fill (ZF), Auto Increment (AI), and a Default Expression of 'NULL'. The other three columns are VARCHAR(255) with a Default Expression of 'NULL'.

Below the table definition, the 'Output' pane shows the results of the drop operation:

#	Time	Action	Message	Duration / Fetch
13	06:27:49	Apply changes to people	Changes applied	
14	06:38:36	Apply changes to people	No changes detected	
15	06:39:02	Apply changes to people	No changes detected	
16	06:40:23	Apply changes to people	No changes detected	
17	06:40:33	Apply changes to people	Changes applied	
18	06:49:59	DROP TABLE 'platzi_blog'.people'	0 row(s) affected	0.015 sec





Nota: las herramientas del DDL se utilizan mayormente en la fase de construcción y mantenimiento de las bases de datos, la manipulación u operación se hace con DML.

Clase 23 DML

DML trata del contenido de la base de datos. Son las siglas de Data Manipulation Language y sus comandos son:

- **Insert:** Inserta o agrega nuevos registros a la tabla. - **Update:** Actualiza o modifica los datos que ya existen.
- **Delete:** Esta sentencia es riesgosa porque puede borrar el contenido de una tabla. - **Select:** Trae información de la base de datos.

Insert

Crearemos este comando en una BD nueva llamada `platzi_test` con la con la estructura del ejercicio anterior

Insert

```
INSERT INTO people (last_name,  
first_name, address, city)  
VALUES ('Hernández', 'Laura',  
'Calle 21', 'Monterrey');
```

En la sección de query realizamos el comando

```
INSERT INTO people(last_name, first_name, address, city)  
VALUES ('Hernandez', 'Laura', 'Calle 21', 'Monterrey');
```

Insertamos los valores con el símbolo de rayo o con ctrl + enter

The screenshot shows the MySQL Workbench interface. In the top-left, the Navigator pane displays the schema `platzi_test` with a single table named `people`. The main area, titled "Query 1 - people - Table", contains the following SQL code:

```

1  INSERT INTO people(last_name, first_name, address, city)
2  VALUES ('Hernandez', 'Laura', 'Calle 21', 'Monterrey');

```

The status bar at the bottom indicates "1 row(s) affected". Below the query results, the "Information" pane shows the execution history for the current session:

#	Time	Action	Message	Duration / Fetch
25	07:20:47	Apply changes to people	Changes applied	
26	07:20:53	INSERT INTO people(last_name, first_name, address, city) VALUES ('He...')	Error Code: 1054. Unknown column 'address' in field list'	0.016 sec
27	07:21:09	Apply changes to people	Changes applied	
28	07:21:15	INSERT INTO people(last_name, first_name, address, city) VALUES ('He...')	Error Code: 1364. Field 'people_id' doesn't have a default value	0.000 sec
29	07:21:25	Apply changes to people	Changes applied	
30	07:21:30	INSERT INTO people(last_name, first_name, address, city) VALUES ('He...')	1 row(s) affected	0.016 sec

The screenshot shows the MySQL Workbench interface. The Navigator pane displays the schema `platzi_test` with the `people` table selected. The main area, titled "Query 1 - people - Table", contains the following SQL code:

```

1 • | SELECT * FROM platzi_test.people;

```

The status bar at the bottom indicates "1 row(s) affected". Below the query results, the "Information" pane shows the execution history for the current session:

#	Time	Action	Message	Duration / Fetch
26	07:20:53	INSERT INTO people(last_name, first_name, address, city) VALUES ('He...')	Error Code: 1054. Unknown column 'address' in field list'	0.016 sec
27	07:21:09	Apply changes to people	Changes applied	
28	07:21:15	INSERT INTO people(last_name, first_name, address, city) VALUES ('He...')	Error Code: 1364. Field 'people_id' doesn't have a default value	0.000 sec
29	07:21:25	Apply changes to people	Changes applied	
30	07:21:30	INSERT INTO people(last_name, first_name, address, city) VALUES ('He...')	1 row(s) affected	0.016 sec
31	07:23:41	SELECT * FROM platzi_test.people LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

Update

```
UPDATE people
SET last_name = 'Chávez', city= 'Mérida'
WHERE person_id = 1;
```

```
UPDATE people
SET first_name = 'Juan'
WHERE city = 'Mérida';
```

```
UPDATE people
SET first_name = 'Juan';
```

comandos

```
UPDATE people SET last_name = 'Chavez', city = 'Merida'
WHERE person_id = 1;
```

MySQL Workbench

Schemas

- platzi_test
 - Tables
 - people
 - Views
 - Stored Procedures
 - Functions
- sys

Query 1 people - Table people

```
1 • UPDATE people SET last_name = 'Chavez', city = 'Merida'
2 WHERE person_id = 1;
```

Information

No object selected

Action Output

#	Time	Action	Message	Duration / Fetch
31	07:23:41	SELECT * FROM platzi_test.people LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
32	07:37:57	INSERT INTO 'platzi_test'.`people`(`people_id`, `last_name`, `first_name`, `address`, `city`) VALUES (4, 'Hernandez', 'Laura', 'Calle 21', 'Monterrey')	1366: Incorrect integer value: "for column 'people_id' at row 1	
33	07:38:43	Apply changes to people	No changes detected	
34	07:40:01	UPDATE people SET last_name = 'Chavez', city = 'Merida' WHERE person_id = 1	Error Code: 1054. Unknown column 'person_id' in 'where clause'	0.000 sec
35	07:40:16	Apply changes to people	Changes applied	
36	07:40:25	UPDATE people SET last_name = 'Chavez', city = 'Merida' WHERE person_id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec

Object Info Session

MySQL Workbench

Schemas

- platzi_test
 - Tables
 - people
 - Views
 - Stored Procedures
 - Functions
- sys

Query 1 people - Table people

```
1 • SELECT * FROM platzi_test.people;
```

Result Grid

people_id	last_name	first_name	address	city
1	Chavez	Israel	reform... reform... reform...	Merida
2	Alanis	Edgar	Avenid... Avenid... Avenid...	Morelia
3	hernandez	Monica	Pensiv... Pensiv... Pensiv...	Aguascal...
4	Hernandez	Laura	Calle 21 NULL	Monterrey NULL
•	NULL	NULL	NULL	NULL

Information

No object selected

people 1

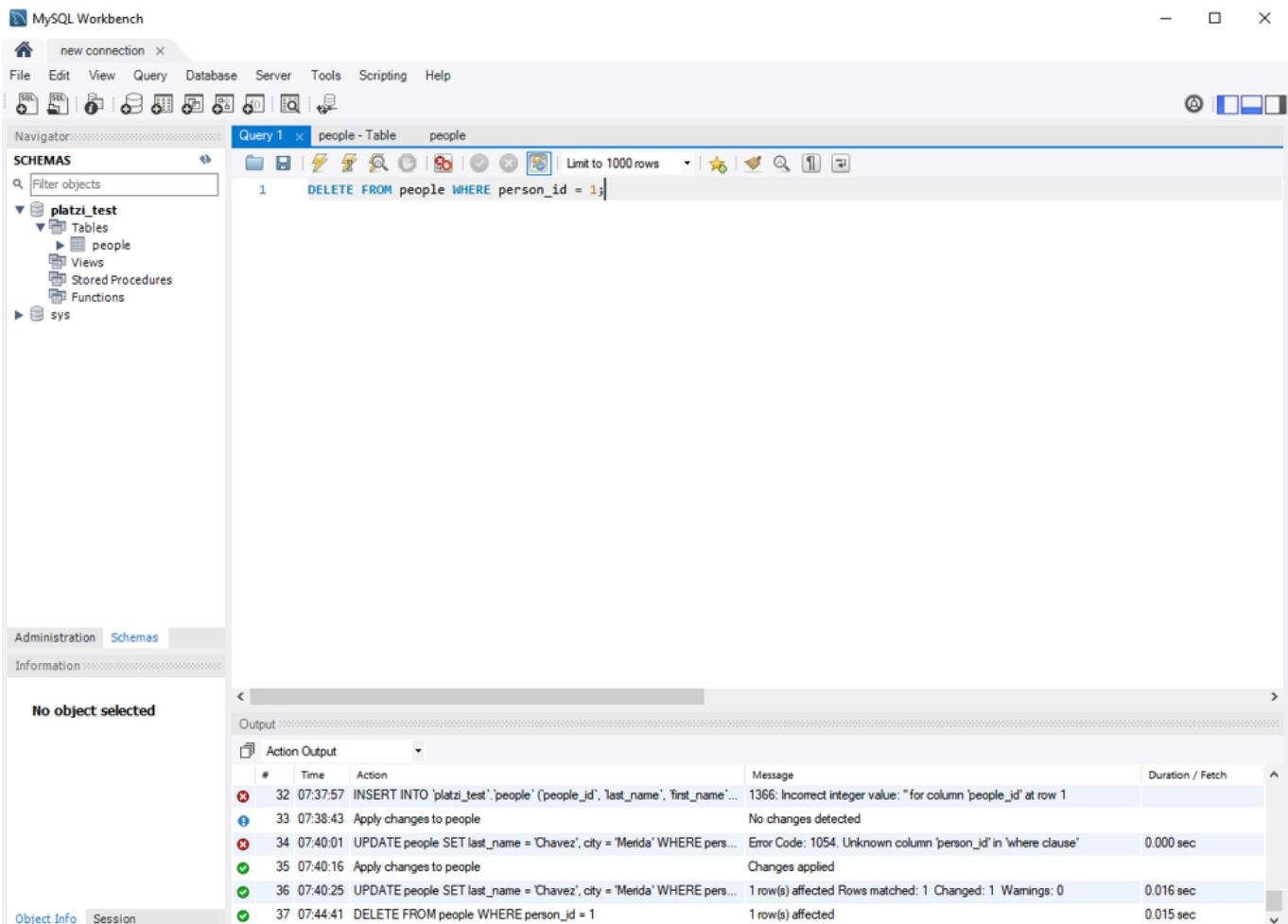
Action Output

#	Time	Action	Message	Duration / Fetch
31	07:23:41	SELECT * FROM platzi_test.people LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
32	07:37:57	INSERT INTO 'platzi_test'.`people`(`people_id`, `last_name`, `first_name`, `address`, `city`) VALUES (4, 'Hernandez', 'Laura', 'Calle 21', 'Monterrey')	1366: Incorrect integer value: "for column 'people_id' at row 1	
33	07:38:43	Apply changes to people	No changes detected	
34	07:40:01	UPDATE people SET last_name = 'Chavez', city = 'Merida' WHERE person_id = 1	Error Code: 1054. Unknown column 'person_id' in 'where clause'	0.000 sec
35	07:40:16	Apply changes to people	Changes applied	
36	07:40:25	UPDATE people SET last_name = 'Chavez', city = 'Merida' WHERE person_id = 1	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec

Object Info Session

Delete

```
DELETE FROM people WHERE person_id = 1;
```



Select

Select

```
SELECT fist_name, last_name
FROM people;
```

```
SELECT first_name, last_name FROM people;
```

The screenshot shows the MySQL Workbench interface. In the top-left, the Navigator pane displays the database schema with a table named 'people' under the 'platzi_test' database. The main area is titled 'Query 1' and contains the SQL query: 'SELECT first_name, last_name FROM people;'. Below the query, the 'Result Grid' shows the following data:

first_name	last_name
Edgar	Alanis
Monica	hernandez
Laura	Hernandez

In the bottom section, the 'Output' pane displays the log of actions taken:

Action	Message	Duration / Fetch
33 07:38:43 Apply changes to people	No changes detected	
34 07:40:01 UPDATE people SET last_name = 'Chavez', city = 'Merida' WHERE pers...	Error Code: 1054. Unknown column 'person_id' in 'where clause'	0.000 sec
35 07:40:16 Apply changes to people	Changes applied	
36 07:40:25 UPDATE people SET last_name = 'Chavez', city = 'Merida' WHERE pers...	1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0	0.016 sec
37 07:44:41 DELETE FROM people WHERE person_id = 1	1 row(s) affected	0.015 sec
38 07:48:11 SELECT first_name, last_name FROM people LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

Clase 24 Que tan standard es SQL

La utilidad más grande de SQL fue unificar la forma en la que pensamos y hacemos preguntas a un repositorio de datos. Ahora que nacen nuevas bases de datos igualmente siguen tomando elementos de SQL.

Practicamos lo anterior como repaso.

```
CREATE TABLE people(
person_id int,
last_name VARCHAR(255),
first_name VARCHAR(255),
address VARCHAR(255),
city VARCHAR(255)
);

INSERT INTO people (last_name, first_name, address, city)
VALUES ('Hernandez', 'Laura', 'Calle 21', 'Monterrey');

.

SELECT first_name, last_name FROM people;
```

```
DROP TABLE people;
```

Realizamos lo mismo en una instancia de postgres gcloud

Tal cual pegamos crear tabla

```
CREATE TABLE people(
person_id int,
last_name VARCHAR(255),
first_name VARCHAR(255),
address VARCHAR(255),
city VARCHAR(255)
);
```

revisamos las tablas con

```
\dt
```

Hacemos las operaciones anteriores

```
INSERT INTO people (last_name, first_name, address, city)
VALUES ('Hernandez', 'Laura', 'Calle 21', 'Monterrey');
.

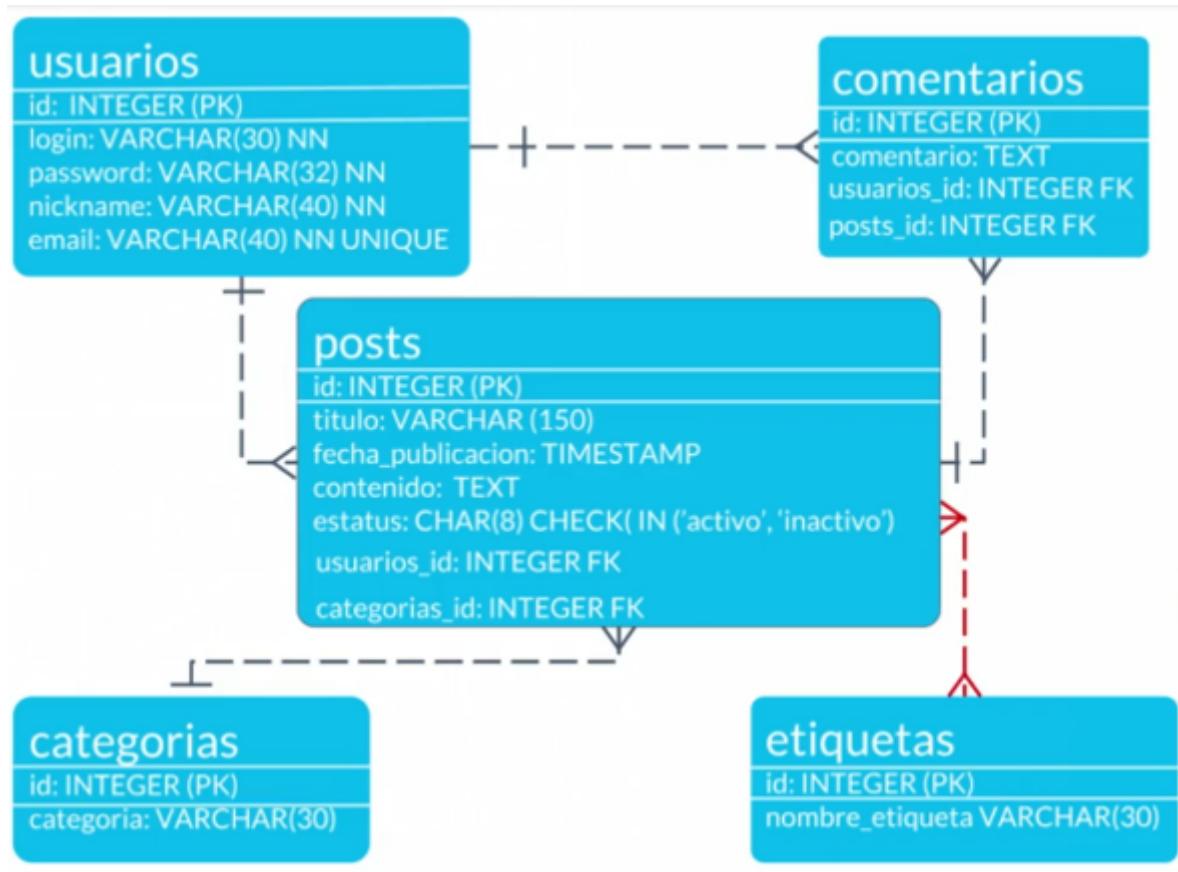
SELECT first_name, last_name FROM people;

DROP TABLE people;
```

El DDL y DML en ambos motores de bases de datos es estandar, cambian solo funciones internas.

Clase 25 Creando Platziblog: tablas independientes

- Una buena práctica es comenzar creando las entidades que no tienen una llave foránea.
- Generalmente en los nombres de bases de datos se evita usar eñes o acentos para evitar problemas en los manejadores de las bases de datos.



Creamos el schema `platziblog` y seleccionamos como default y creamos las siguientes tablas

- categorías

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help' are visible. The 'Navigator' pane on the left lists 'SCHEMAS' (platzi_test, platziblog, sys), 'Tables' (people), 'Views', 'Stored Procedures', 'Functions', and 'Information'. The main workspace shows 'Query 1' with the following table definition:

```

Table Name: categorias Schema: platziblog
Charset/Collation: Default Charset Default Collation Engine: InnoDB
Comments:

Column Name Datatype PK NN UQ B UN ZF AI G Default/Expression
id INT ✓ ✓
nombre_categoria VARCHAR(30) ✓ ✓

```

Below the table definition, a detailed view of the 'nombre_categoria' column is shown:

```

Column Name: nombre_categoria Data Type: VARCHAR(30)
Charset/Collation: Default Charset Default Collation
Comments:
Storage: Virtual Primary Key Not Null Unique
Binary Unsigned Zero Fill
Auto Increment Generated

```

The 'Output' pane at the bottom displays a history of actions:

#	Time	Action	Message	Duration / Fetch
37	07:44:41	DELETE FROM people WHERE person_id = 1	1 row(s) affected	0.015 sec
38	07:48:11	SELECT first_name, last_name FROM people LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec
39	11:48:47	Apply changes to platziblog	Changes applied	
40	11:49:08	Apply changes to platziblog	Changes applied	
41	11:49:21	Apply changes to platziblog	No changes detected	
42	11:49:47	DROP DATABASE platziblog	0 row(s) affected	0.000 sec



Review the SQL Script to be Applied on the Database

Online DDL

Algorithm:	Default	Lock Type:	Default
------------	---------	------------	---------

```

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

```

Buttons at the bottom: Back, Apply, Cancel.

Creamos de igual forma la tabla Etiquetas

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help' are visible. The 'Database' tab is selected. The main workspace shows the 'etiquetas - Table' configuration. The 'Table Name' is set to 'etiquetas', 'Schema' to 'platziblog', 'Charset/Collation' to 'Default Charset', and 'Engine' to 'InnoDB'. The 'Comments' field is empty. Below this, the table structure is defined with two columns: 'Id' (INT, PK, NN) and 'nombre_etiqueta' (VARCHAR(45)). The 'Storage' section includes options for Virtual, Primary Key, Binary, Auto Increment, and others. The 'Output' section at the bottom shows a log of actions, including applying changes to 'categorias', 'etiquetas', and 'usuarios', dropping the 'platzi_test' database, and applying changes to 'etiquetas'. The 'Object Info' and 'Session' tabs are also visible.

Apply SQL Script to Database



Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

Algorithm:

Default

Lock Type:

Default

```
1  CREATE TABLE `platziblog`.`etiquetas` (
2      `id` INT NOT NULL AUTO_INCREMENT,
3      `nombre_etiqueta` VARCHAR(45) NULL,
4      PRIMARY KEY (`id`));
5
```



Back

Apply

Cancel

Tabla Usuarios

The screenshot shows the MySQL Workbench interface. In the top left, the Navigator pane displays the schema structure under 'platziblog'. The main area is titled 'Query 1' and shows the creation of a new table named 'usuarios'. The table definition includes columns: id (INT, PK, AI), login (VARCHAR(30)), password (VARCHAR(32)), nickname (VARCHAR(40)), and email (VARCHAR(40)). Below this, the 'Output' pane shows the log of actions taken:

#	Time	Action	Message	Duration / Fetch
44	12:04:18	Apply changes to etiquetas	Changes applied	
45	12:23:25	Apply changes to usuarios	Changes applied	
46	12:35:26	DROP DATABASE 'platzi_test'	3 row(s) affected	0.047 sec
47	12:35:38	Apply changes to usuarios	No changes detected	
48	12:37:18	Apply changes to etiquetas		
49	12:37:40	Apply changes to etiquetas	Changes applied	

Apply SQL Script to Database



Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

Algorithm:

Default

Lock Type:

Default

```
1  CREATE TABLE `platziblog`.`usuarios` (
2      `id` INT NOT NULL,
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));
9
```



Back

Apply

Cancel

Clase 26 Creando Platziblog: tablas transitivas

Creando Platziblog: tablas dependientes El comando “**cascade**” sirve para que cada que se haga un update en la tabla principal, se refleje también en la tabla en la que estamos creando la relación.

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator: post - Table post - Table post - Table

SCHEMAS

platziblog

- Tables: categorias, etiquetas, usuarios
- Views
- Stored Procedures
- Functions

sys

Table Name: post Schema: platziblog Charset/Collation: utf8 utf8_bin Engine: InnoDB

Comments:

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G	Default/Expression
id	INT	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
titulo	VARCHAR(130)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
fecha_publicacion	TIMESTAMP	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
contenido	TEXT	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
estatus	CHAR(8)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	'activo'
usuario_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
categoria_id	INT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Column Name: Data Type: Default:

Charset/Collation: Default Charset Default Collation

Comments:

Storage: Virtual Stored
Primary Key Not Null Unique
Binary Unsigned Zero Fill
Auto Increment Generated

Columns Indexes Foreign Keys Triggers Partitioning Options

Schema: platziblog

Action Output

#	Time	Action	Message	Duration / Fetch
56	12:54:50	Apply changes to post	Error 1146: Table 'platziblog post' doesn't exist SQL ...	
57	12:55:27	Apply changes to post	Error 1146: Table 'platziblog post' doesn't exist SQL ...	
58	12:55:46	Apply changes to post	Error 1146: Table 'platziblog post' doesn't exist SQL ...	
59	12:56:07	Apply changes to post	Error 1146: Table 'platziblog post' doesn't exist SQL ...	
60	12:56:54	Apply changes to post	Error 1146: Table 'platziblog post' doesn't exist SQL ...	

Object Info Session

Apply SQL Script to Database



Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

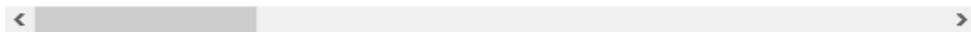
Algorithm:

Default

Lock Type:

Default

```
1  CREATE TABLE `platziblog`.`post` (
2      `id` INT NOT NULL AUTO_INCREMENT,
3      `titulo` VARCHAR(130) NULL,
4      `fecha_publicacion` TIMESTAMP NULL,
5      `contenido` TEXT NOT NULL,
6      `estatus` CHAR(8) NULL DEFAULT 'activo',
7      `usuario_id` INT NOT NULL,
8      `categoria_id` INT NOT NULL,
9      PRIMARY KEY (`id`)
10     ENGINE = InnoDB
11     DEFAULT CHARACTER SET = utf8;
12
```



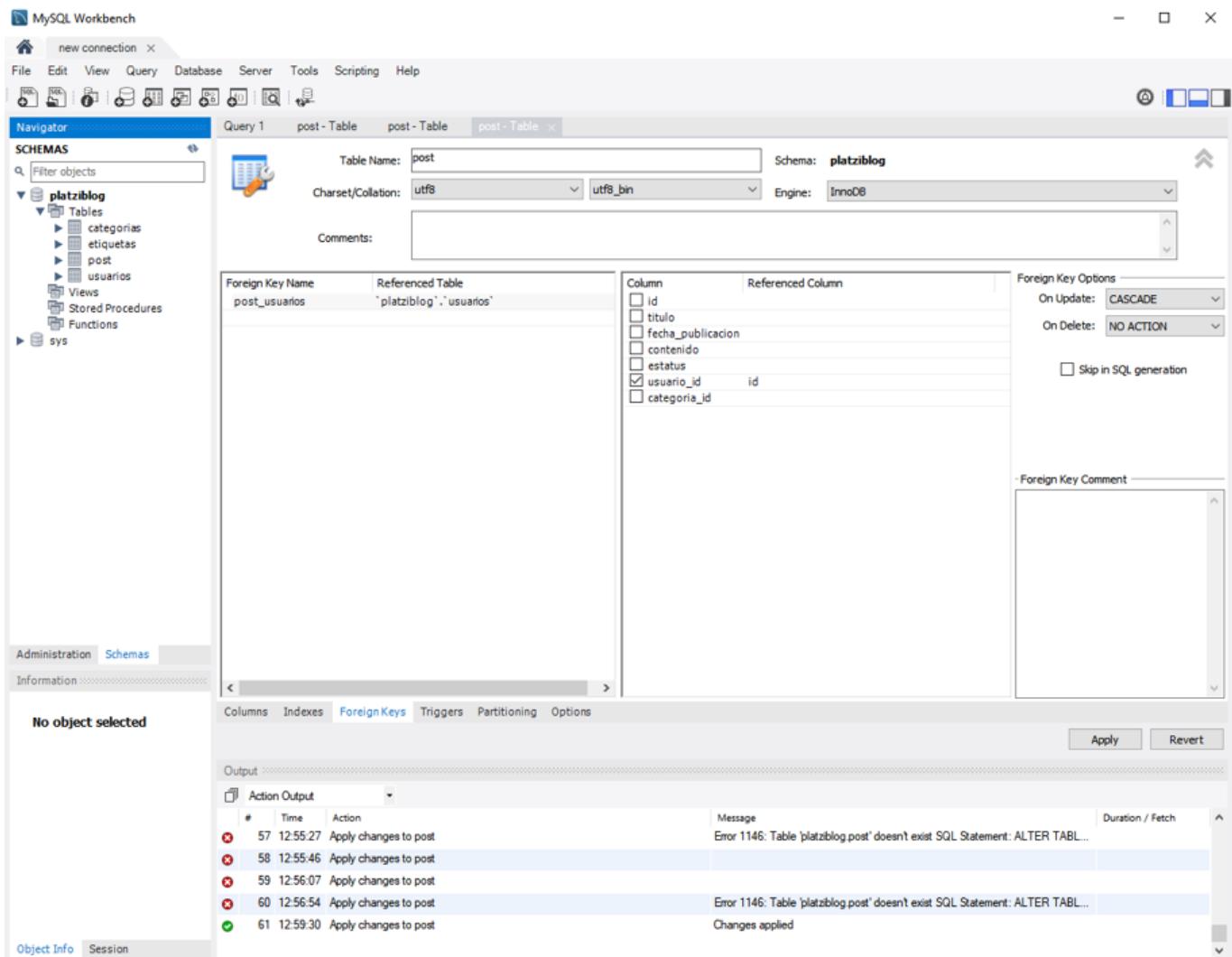
Back

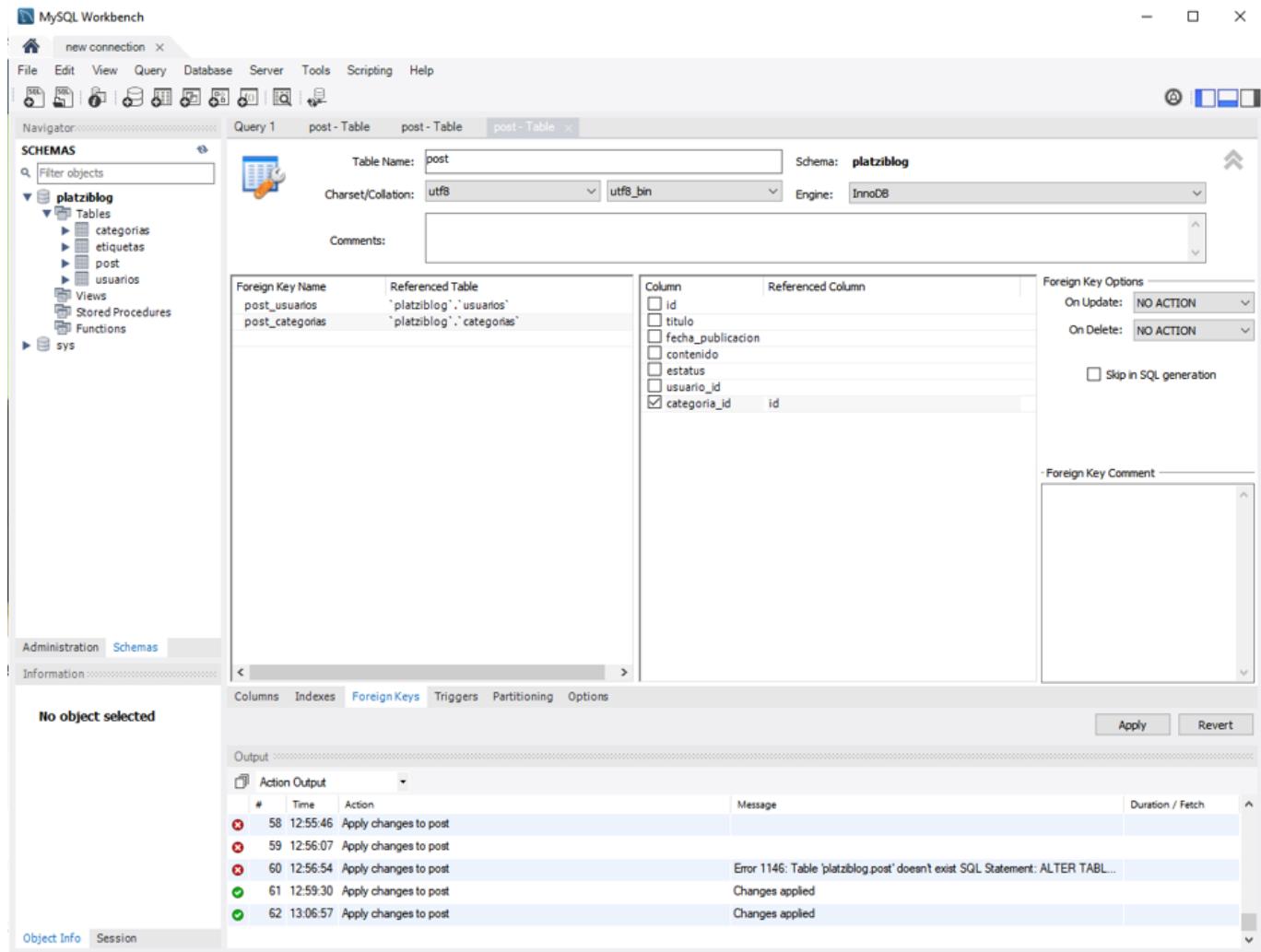
Apply

Cancel

Vamos a la pestaña Foreign Key y asignamos las relaciones.

Asignamos la llave foranea del usuario_id





Apply SQL Script to Database



Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

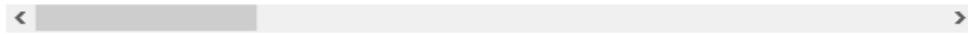
Algorithm:

Default

Lock Type:

Default

```
1   ALTER TABLE `platziblog`.`post`
2     ADD INDEX `post_categorias_idx` (`categoria_id` ASC);
3   ;
4   ALTER TABLE `platziblog`.`post`
5     ADD CONSTRAINT `post_categorias`
6       FOREIGN KEY (`categoria_id`)
7         REFERENCES `platziblog`.`categorias` (`id`)
8         ON DELETE NO ACTION
9         ON UPDATE NO ACTION;
10
```



Back

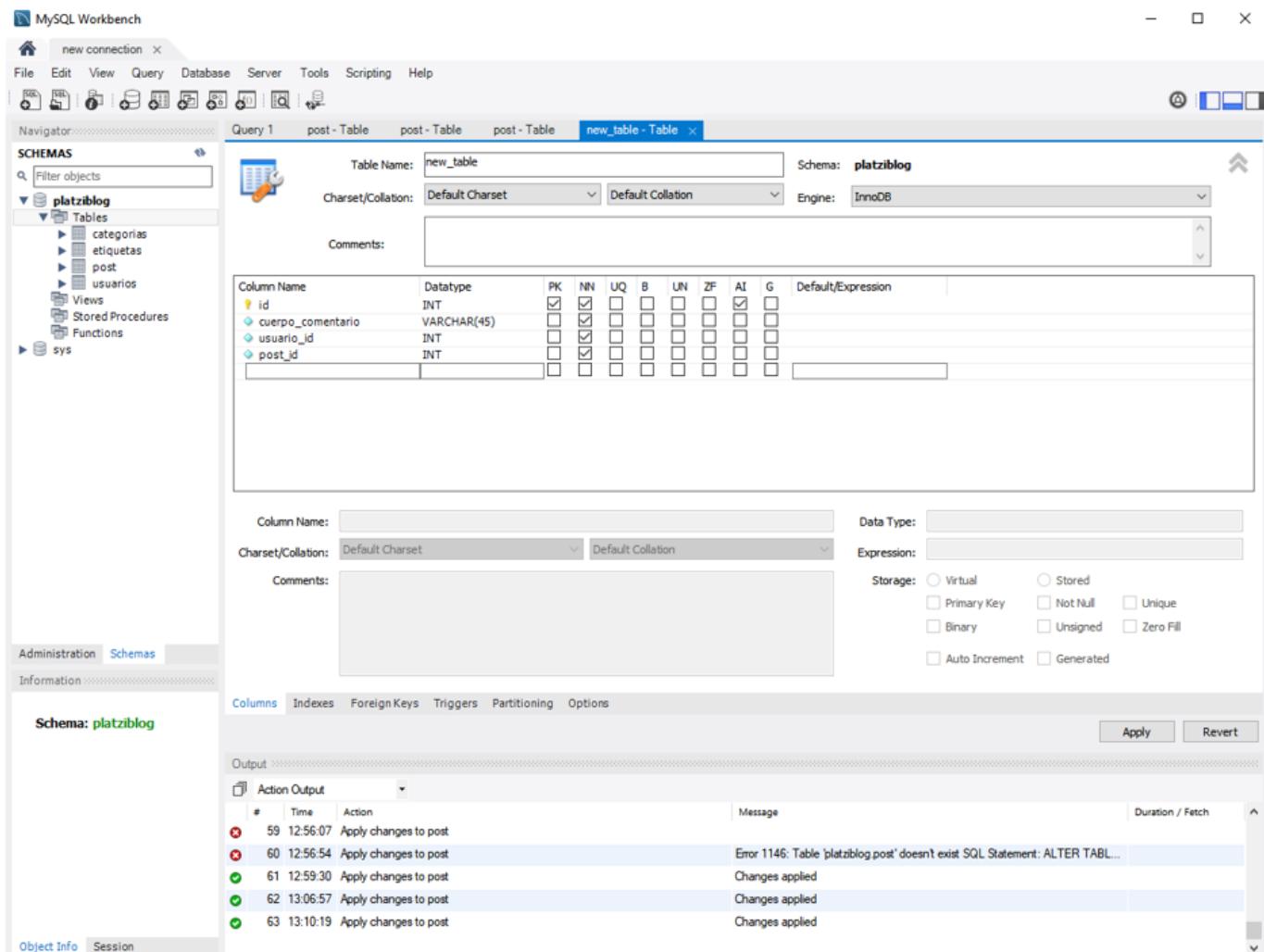
Apply

Cancel

Clase 27 Creando Platziblog: tablas transitivas

- Las tablas transitivas sirven como puente para unir dos tablas. No tienen contenido semántico.
- **Reverse Engineer** nos reproduce el esquema del cual nos basamos para crear nuestras tablas. Es útil cuando llegas a un nuevo trabajo y quieres entender cuál fue la mentalidad que tuvieron al momento de crear las bases de datos.

Creamos la tabla comentarios.



Apply SQL Script to Database x

Review SQL Script Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

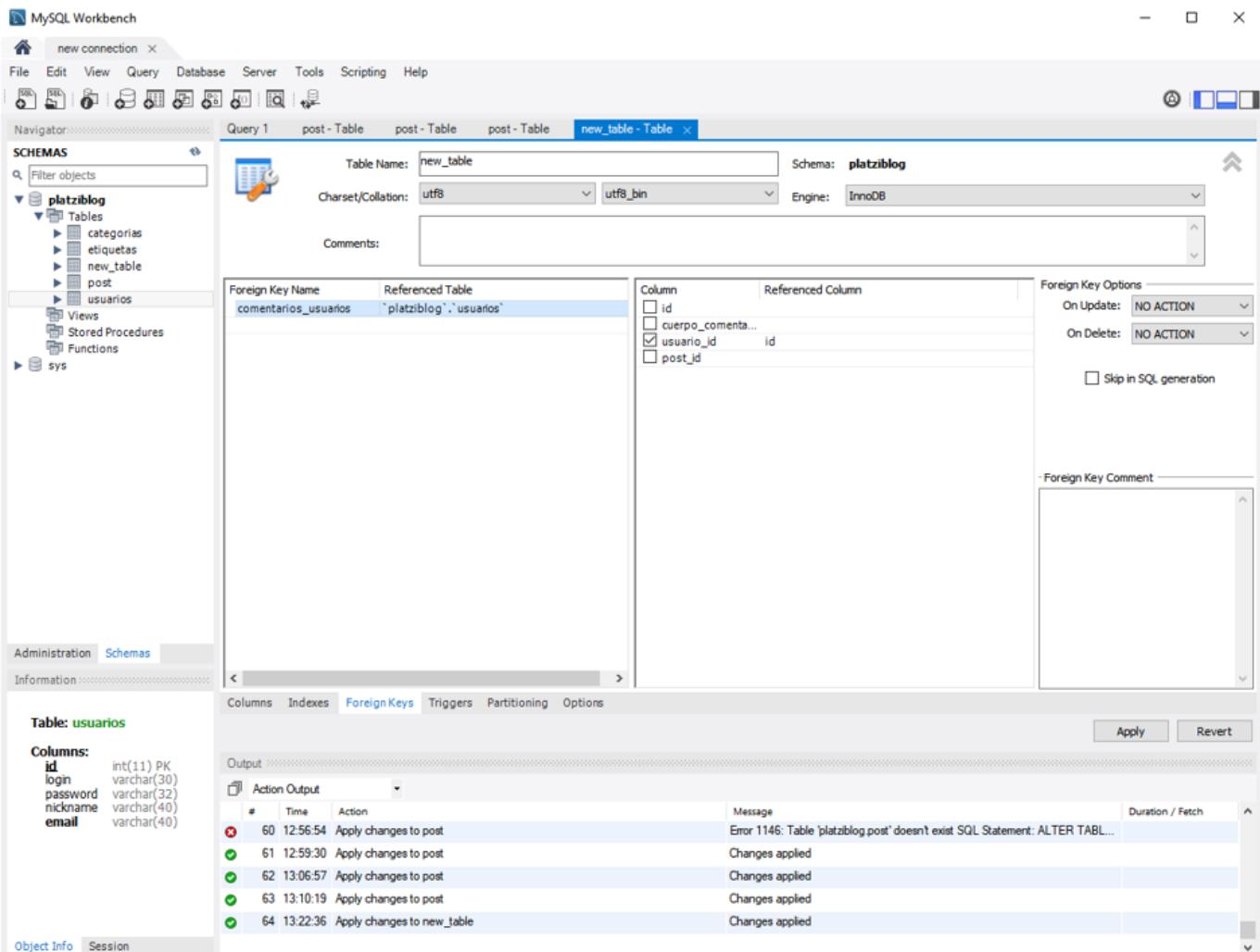
Algorithm: Default Lock Type: Default

```
1  CREATE TABLE `platziblog`.`new_table` (
2      `id` INT NOT NULL AUTO_INCREMENT,
3      `cuerpo_comentario` VARCHAR(45) NOT NULL,
4      `usuario_id` INT NOT NULL,
5      `post_id` INT NOT NULL,
6      PRIMARY KEY (`id`));
7
```

< >

Back Apply Cancel

Creamos las llaves foráneas para los comentarios de los usuarios



Apply SQL Script to Database



Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

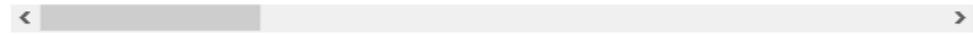
Online DDL

Algorithm: Default

Lock Type:

Default

```
1   ALTER TABLE `platziblog`.`new_table`  
2     ADD INDEX `comentarios_usuarios_idx` (`usuario_id` ASC);  
3   ;  
4   ALTER TABLE `platziblog`.`new_table`  
5     ADD CONSTRAINT `comentarios_usuarios`  
6       FOREIGN KEY (`usuario_id`)  
7         REFERENCES `platziblog`.`usuarios` (`id`)  
8         ON DELETE NO ACTION  
9         ON UPDATE NO ACTION;  
10
```

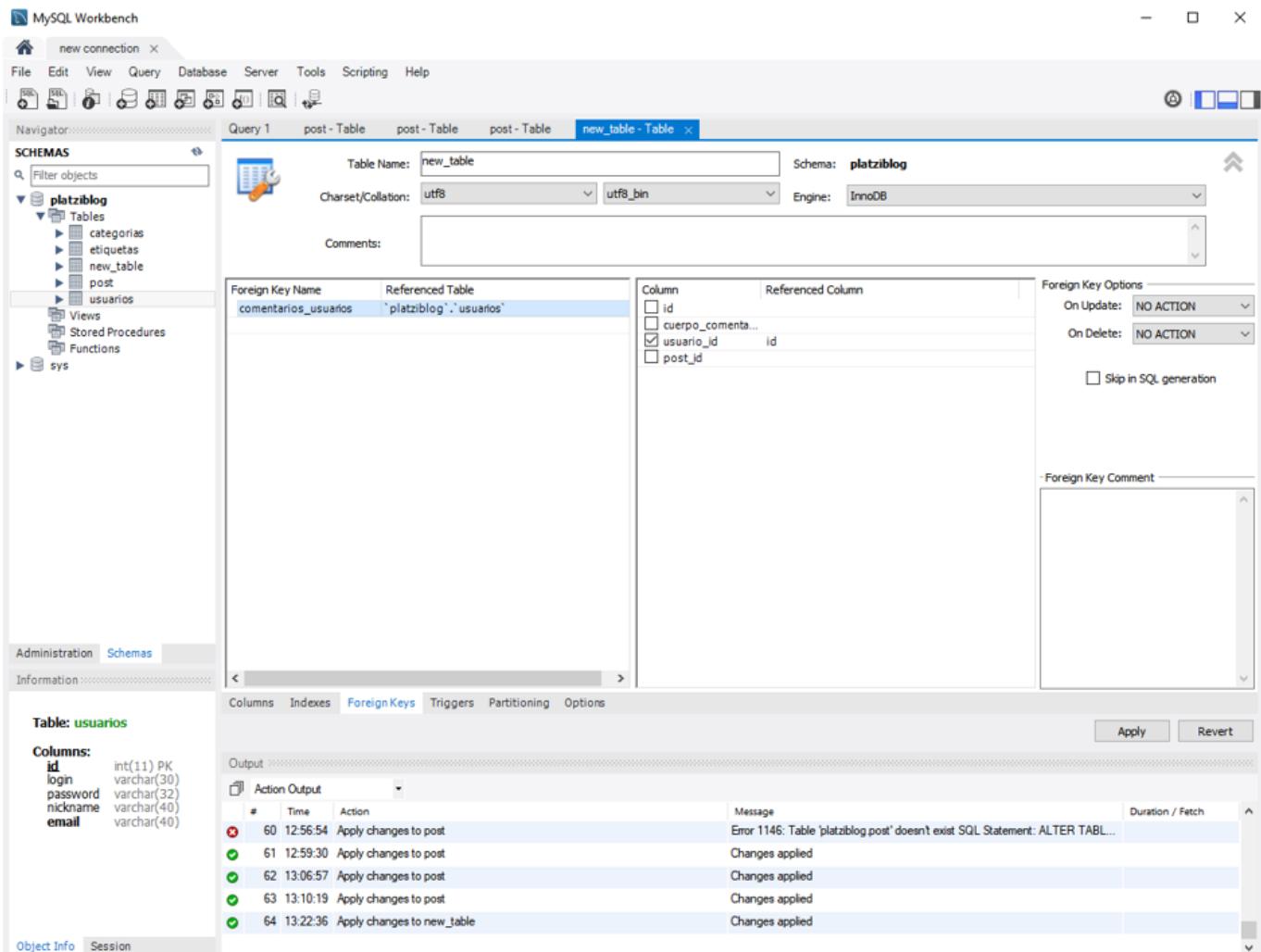


Back

Apply

Cancel

Creamos las llaves foráneas para los comentarios de los posts



Apply SQL Script to Database

Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

Algorithm: Default

Lock Type: Default

```
1  ALTER TABLE `platziblog`.`new_table`  
2    ADD INDEX `comentarios_usuarios_idx` (`usuario_id` ASC);  
3 ;  
4  ALTER TABLE `platziblog`.`new_table`  
5    ADD CONSTRAINT `comentarios_usuarios`  
6      FOREIGN KEY (`usuario_id`)  
7        REFERENCES `platziblog`.`usuarios` (`id`)  
8        ON DELETE NO ACTION  
9        ON UPDATE NO ACTION;  
10
```

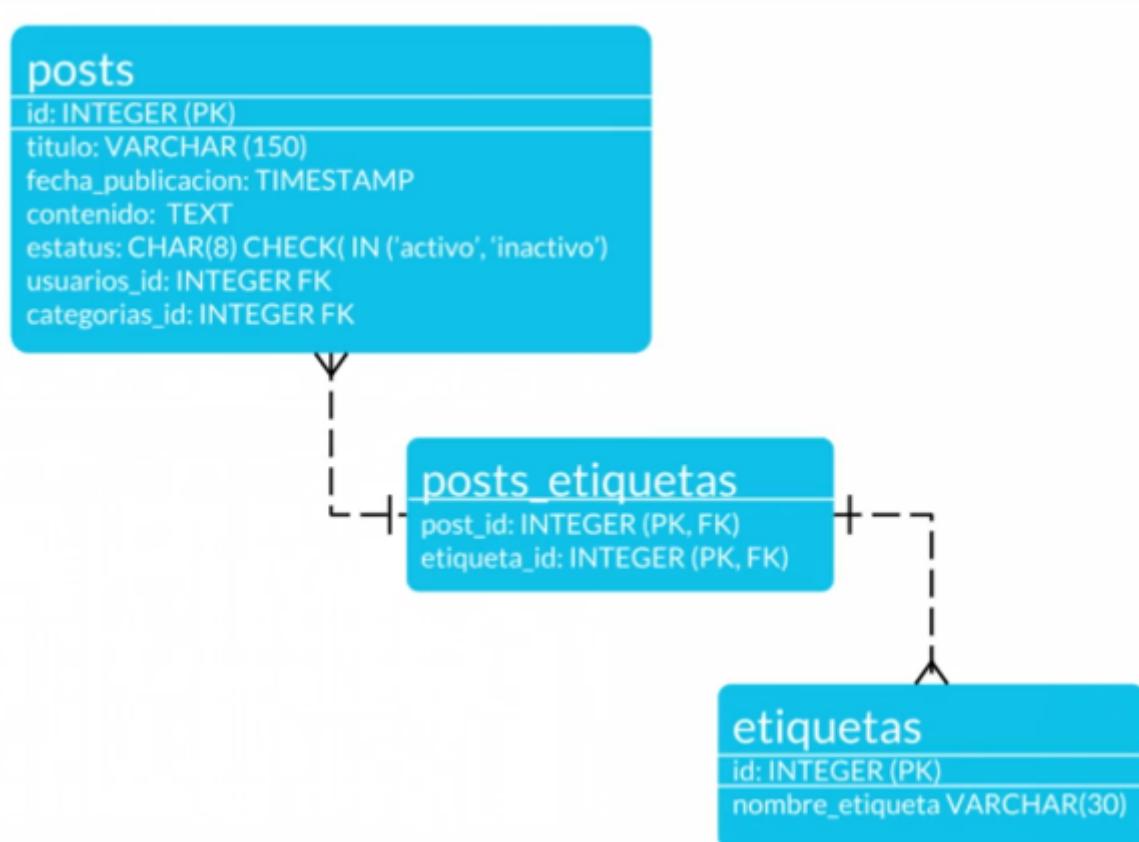
< >

Back

Apply

Cancel

Ahora solo nos queda crear la tabla intermedia para romper la relacion muchos a muchos.



Para ello creamos la tabla intermedia post_etiquetas

The screenshot shows the MySQL Workbench interface. In the top navigation bar, 'File', 'Edit', 'View', 'Query', 'Database', 'Server', 'Tools', 'Scripting', and 'Help' are visible. Below the navigation bar, the 'Navigator' pane shows the database schema with 'platziblog' selected. Under 'Tables', 'post_etiquetas' is listed. The main workspace is titled 'Query 1' and shows the creation of the 'post_etiquetas' table. The table definition is as follows:

```
Table Name: post_etiquetas
Schema: platziblog
Charset/Collation: utf8 utf8_bin
Engine: InnoDB
```

The 'Foreign Keys' tab is selected, showing a foreign key named 'postetiquetas_post' referencing the 'post' table. The foreign key options are set to 'NO ACTION' for both 'On Update' and 'On Delete'. The output pane at the bottom shows the execution log:

#	Time	Action	Message	Duration / Fetch
70	16:42:35	Apply changes to post_etiquetas	Changes applied	
71	16:43:26	Apply changes to post_etiquetas	No changes detected	
72	16:43:32	Apply changes to post_etiquetas	No changes detected	
73	16:43:33	Apply changes to post_etiquetas	Changes applied	
74	16:44:35	Apply changes to post_etiquetas	Changes applied	

Apply SQL Script to Database



Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

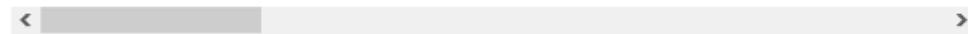
Algorithm:

Default

Lock Type:

Default

```
1   ALTER TABLE `platziblog`.`post_etiquetas`  
2     ADD INDEX `postetiquetas_post_idx` (`post_id` ASC);  
3   ;  
4   ALTER TABLE `platziblog`.`post_etiquetas`  
5     ADD CONSTRAINT `postetiquetas_post`  
6       FOREIGN KEY (`post_id`)  
7         REFERENCES `platziblog`.`post` (`id`)  
8         ON DELETE NO ACTION  
9         ON UPDATE NO ACTION;  
10
```

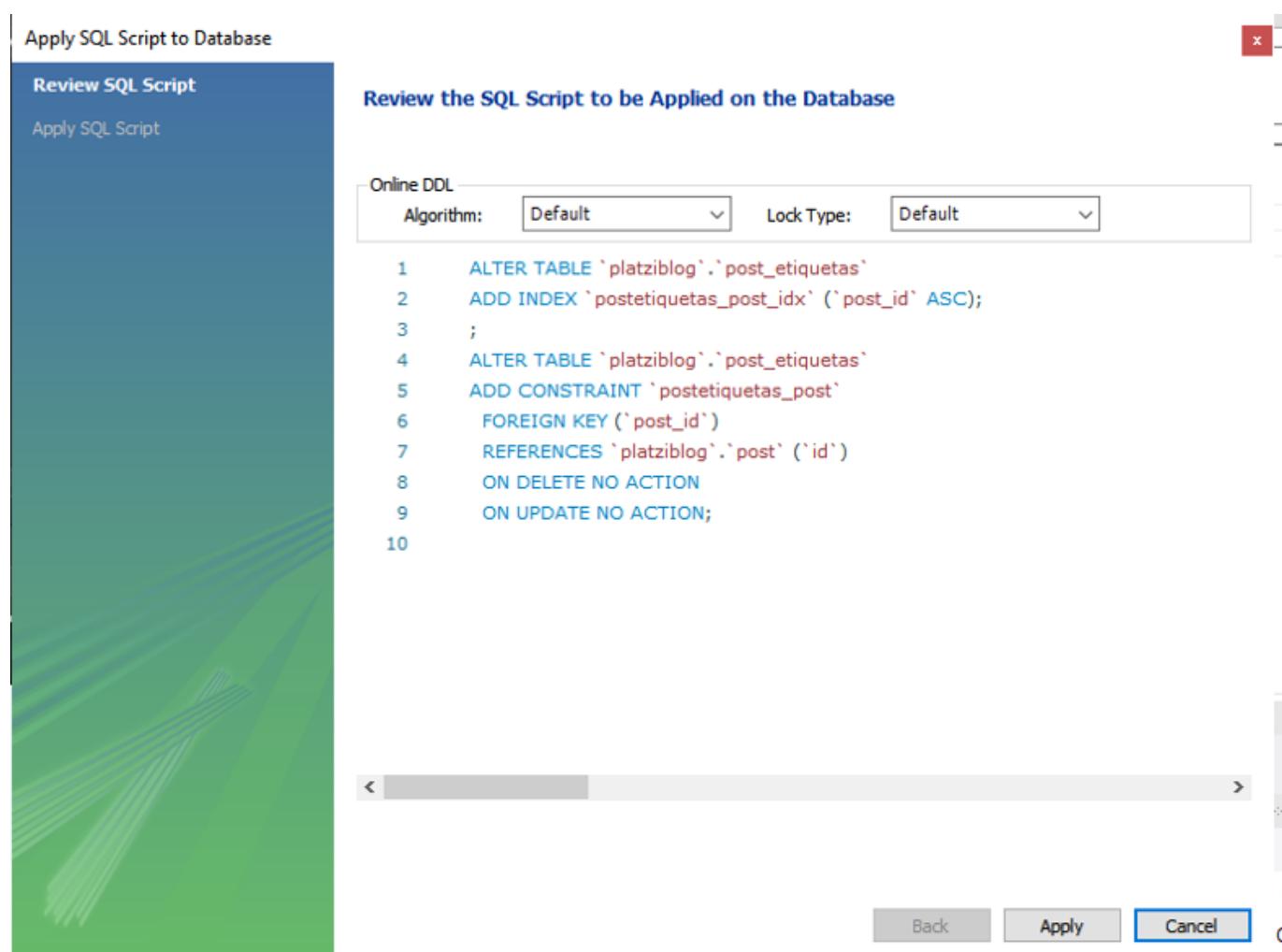
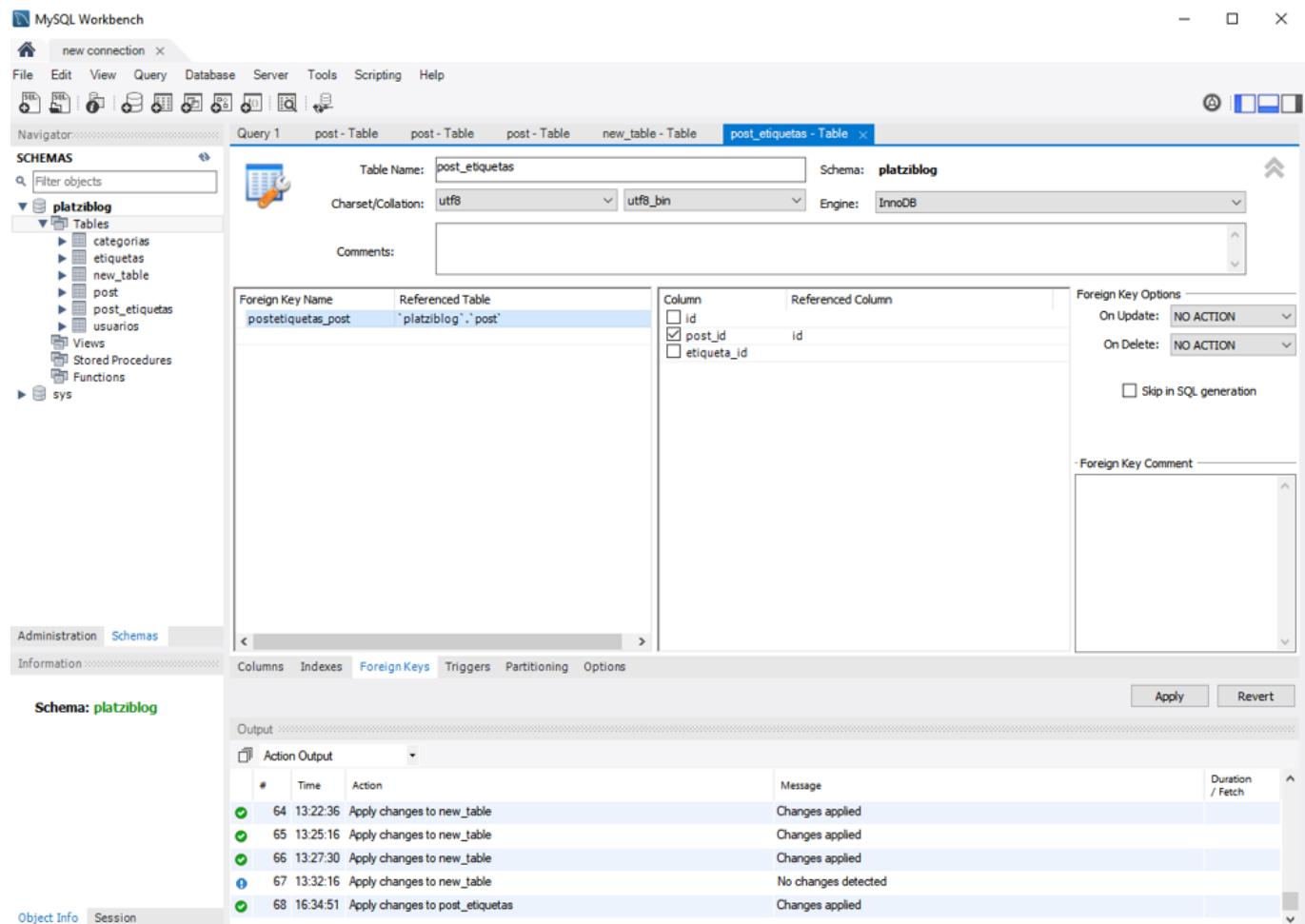


Back

Apply

Cancel

Ligamos las llaves foráneas para post_id



Ligamos las llaves foráneas para etiquetas_id

The screenshot shows the MySQL Workbench interface for creating a foreign key constraint. The left sidebar shows the schema structure under 'platziblog'. The main panel is titled 'post_etiquetas - Table' and contains the following details:

- Table Name:** post_etiquetas
- Schema:** platziblog
- Charset/Collation:** utf8 / utf8_bin
- Engine:** InnoDB
- Comments:** (empty)
- Foreign Key Name:** postetiquetas_post
- Referenced Table:** 'platziblog'.post
- Column:** id
- Referenced Column:** post_id
- Foreign Key Options:**
 - On Update: NO ACTION
 - On Delete: NO ACTION
 - Skip in SQL generation
- Foreign Key Comment:** (empty)

Below the table definition, the 'Foreign Keys' tab is selected in the 'Information' section. The 'Output' pane shows the following log entries:

#	Time	Action	Message	Duration / Fetch
71	16:43:26	Apply changes to post_etiquetas	Changes applied	
72	16:43:32	Apply changes to post_etiquetas	No changes detected	
73	16:43:33	Apply changes to post_etiquetas	No changes detected	
74	16:44:35	Apply changes to post_etiquetas	Changes applied	
75	16:45:28	Apply changes to post_etiquetas	Changes applied	

Apply SQL Script to Database x

Review SQL Script X

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

Algorithm: Default Lock Type: Default

```
1   ALTER TABLE `platziblog`.`post_etiquetas`  
2     ADD INDEX `postetiquetas_etiquetas_idx` (`etiqueta_id` ASC);  
3   ;  
4   ALTER TABLE `platziblog`.`post_etiquetas`  
5     ADD CONSTRAINT `postetiquetas_etiquetas`  
6       FOREIGN KEY (`etiqueta_id`)  
7         REFERENCES `platziblog`.`etiquetas` (`id`)  
8           ON DELETE NO ACTION  
9             ON UPDATE NO ACTION;  
10
```

< >

Back Apply Cancel

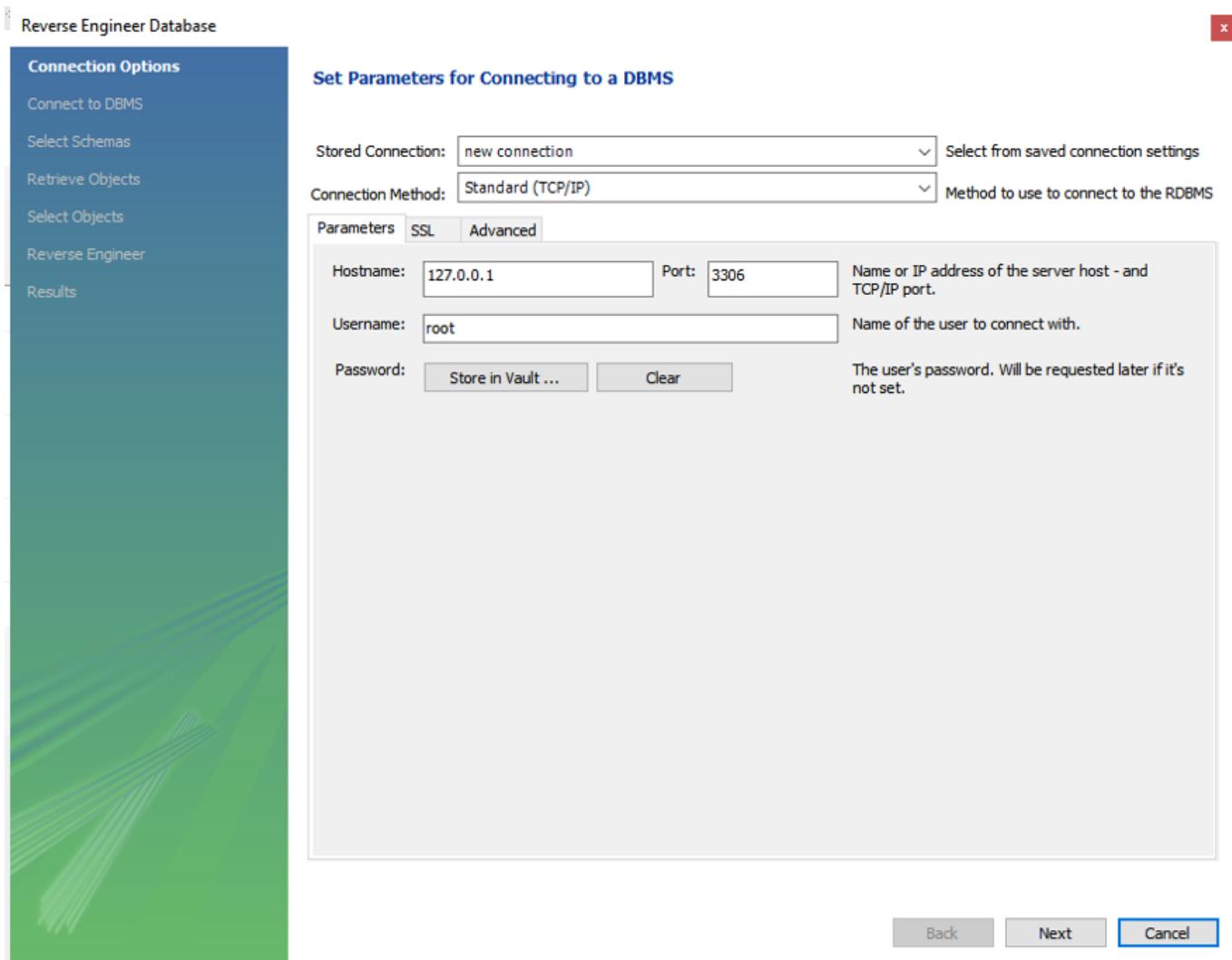


Reverse Engineer

Seleccionamos la opción Database para acceder a reverse engineer

The screenshot shows the MySQL Workbench interface with the 'Database' tab selected in the top menu bar. A context menu is open over the 'post_etiquetas' table in the 'platziblog' schema. The 'Reverse Engineer...' option is highlighted. The main workspace displays the 'Foreign Keys' tab for the 'post_etiquetas' table. It lists two foreign keys: 'postetiquetas_etiquetas' referencing 'etiquetas' and 'postetiquetas_post' referencing 'post'. The 'Column' column shows 'post_id' and 'etiqueta_id' respectively. On the right, 'Foreign Key Options' are set to 'NO ACTION' for both update and delete operations. Below the table list, an 'Output' section titled 'Action Output' shows a log of recent actions:

#	Time	Action	Message	Duration / Fetch
72	16:43:32	Apply changes to post_etiquetas	No changes detected	
73	16:43:33	Apply changes to post_etiquetas	No changes detected	
74	16:44:35	Apply changes to post_etiquetas	Changes applied	
75	16:45:28	Apply changes to post_etiquetas	Changes applied	
76	16:51:06	Apply changes to post_etiquetas	Changes applied	



Reverse Engineer Database

Connection Options

Connect to DBMS

Select Schemas

Retrieve Objects

Select Objects

Reverse Engineer

Results

Connect to DBMS and Fetch Information

The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.

Connect to DBMS
 Retrieve Schema List from Database
 Check Common Server Configuration Issues

Connecting to DBMS...

MySQL Workbench

Please enter password for the following service:

Service: Mysql@127.0.0.1:3306

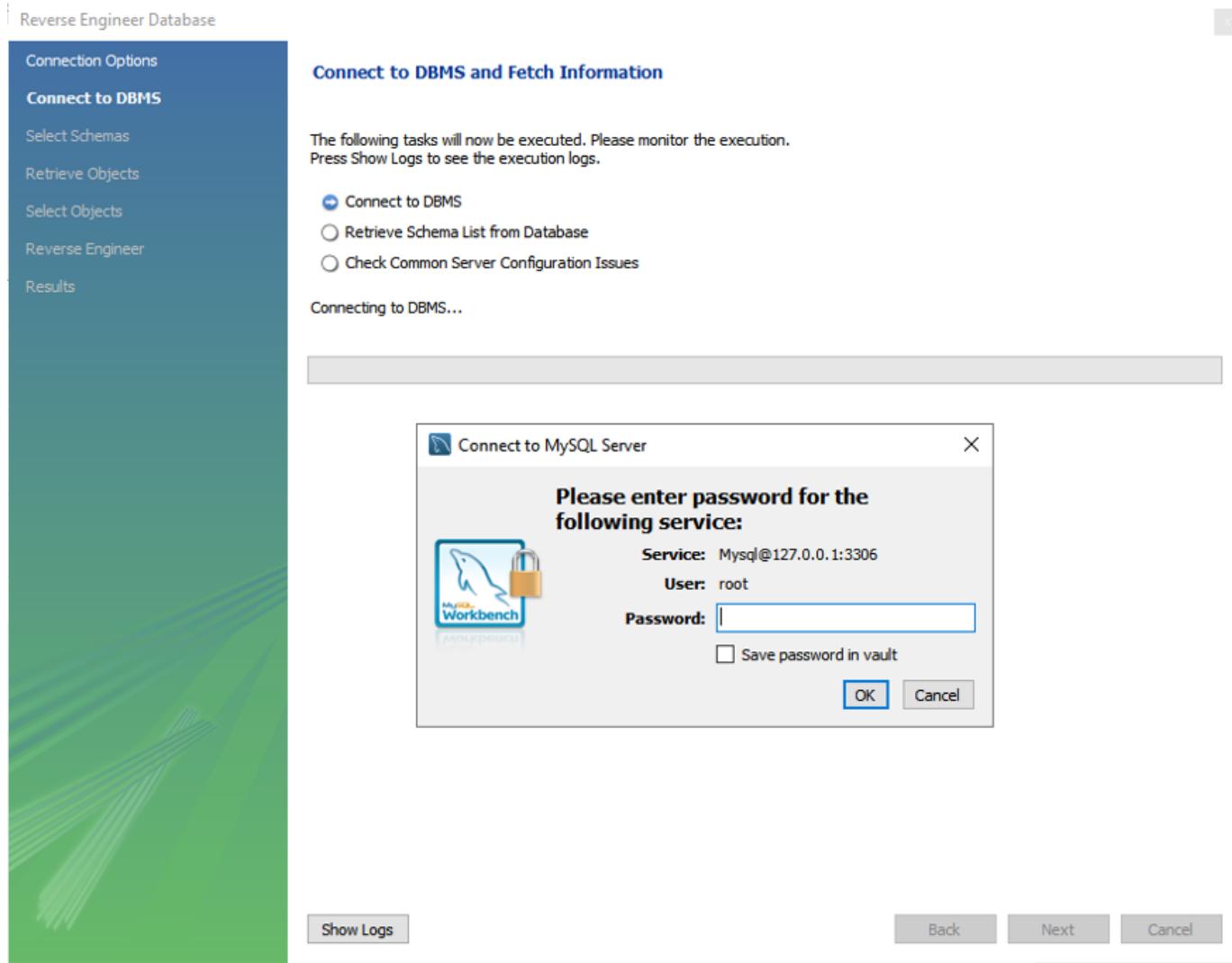
User: root

Password:

Save password in vault

OK Cancel

Show Logs Back Next Cancel



This screenshot shows the MySQL Workbench interface. On the left, there's a sidebar with various options like 'Connection Options', 'Select Schemas', etc. The main area is titled 'Connect to DBMS and Fetch Information'. It displays a message about executing tasks and a progress bar. A modal window titled 'Connect to MySQL Server' is open, prompting for a password for the 'root' user at 'Mysql@127.0.0.1:3306'. The MySQL Workbench logo is visible in the background. At the bottom, there are buttons for 'Show Logs', 'Back', 'Next', and 'Cancel'.

Reverse Engineer Database X

Connection Options
Connect to DBMS
Select Schemas
Retrieve Objects
Select Objects
Reverse Engineer
Results

Select Schemas to Reverse Engineer

 **Select the schemas you want to include:**

platziblog

Back Next Cancel

Reverse Engineer Database X

Connection Options
Connect to DBMS
Select Schemas
Retrieve Objects
Select Objects
Reverse Engineer
Results

Retrieve and Reverse Engineer Schema Objects

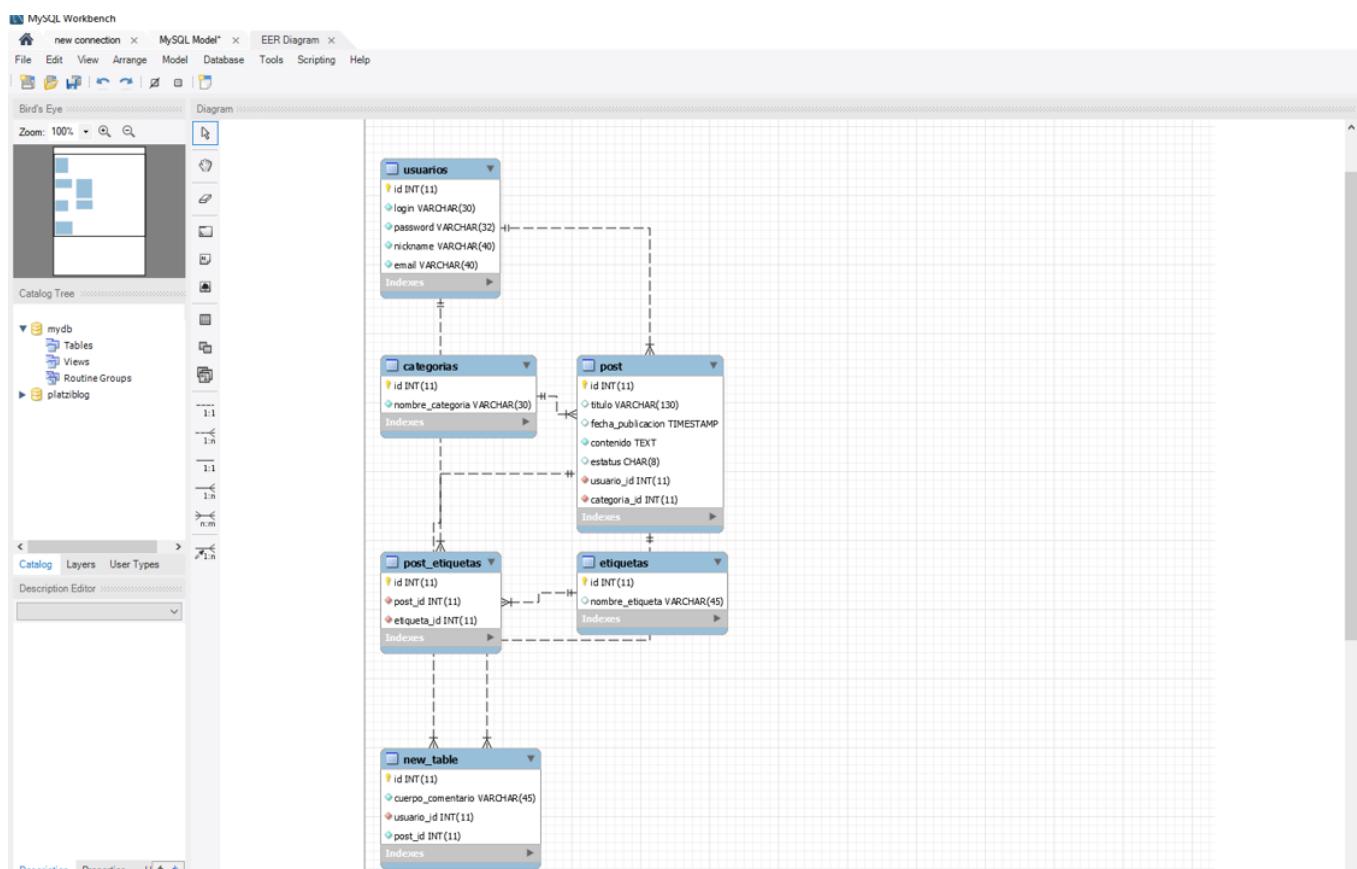
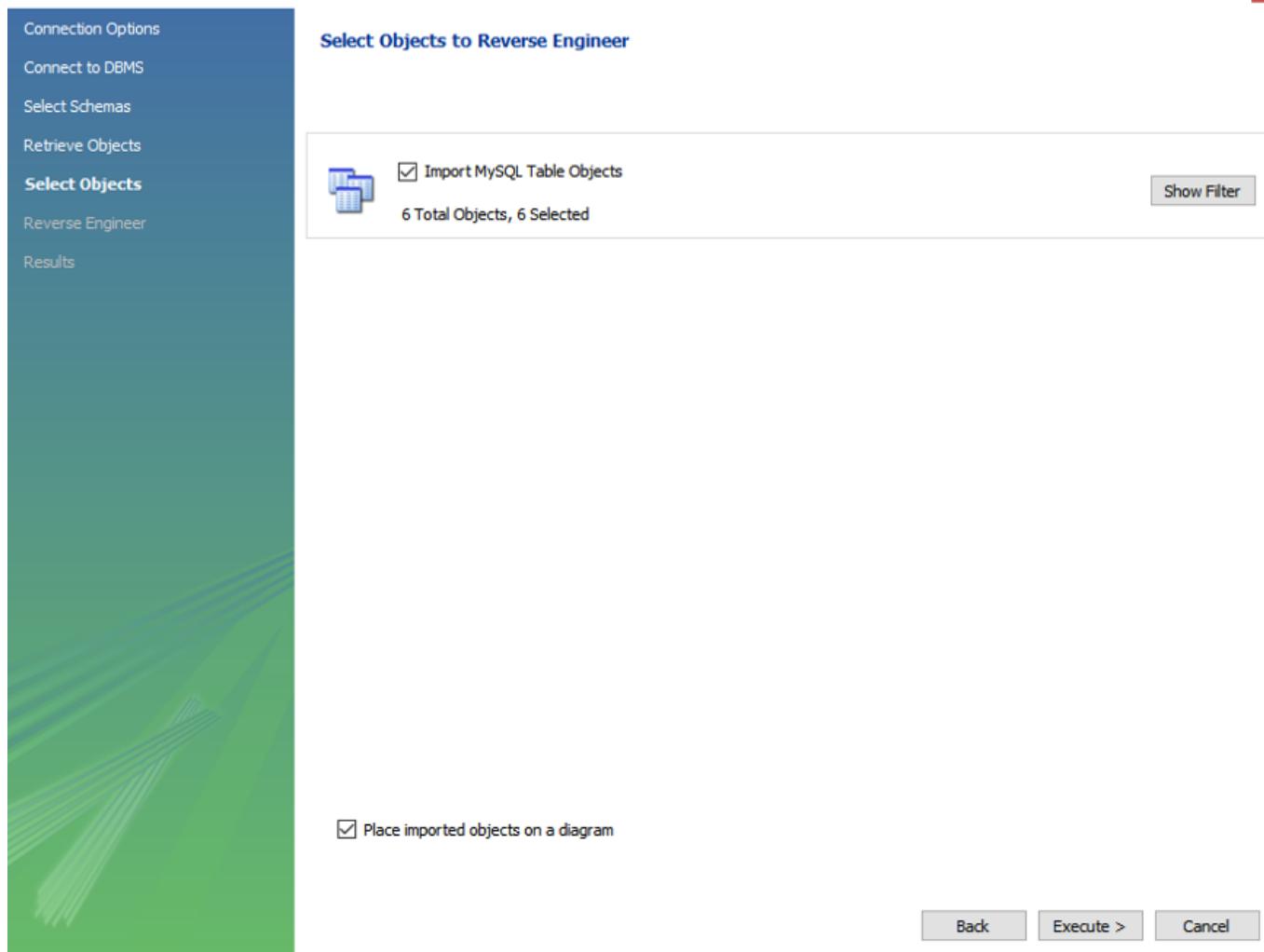
The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.

Retrieve Objects from Selected Schemas
 Check Results

Retrieval Completed Successfully
Finished.

Show Logs Back Next Cancel

Reverse Engineer Database



Clase 28 Por qué las consultas son tan importantes

Las consultas o queries a una base de datos son una parte fundamental ya que esto podría salvar un negocio o empresa. Alrededor de las consultas a las bases de datos se han creado varias especialidades como ETL o transformación de datos, business intelligence e incluso machine learning.

Clase 29 Estructura básica de un Query

Los queries son la forma en la que estructuramos las preguntas que se harán a la base de datos. Transforma preguntas en sintaxis.

El query tiene básicamente 2 partes: SELECT y FROM y puede aparecer una tercera como WHERE.

La estrellita o asterisco (*) quiere decir que vamos a seleccionar todo sin filtrar campos.

Primer Consulta

```
SELECT*
FROM posts
WHERE fecha_publicacion < '2024'
```

Clase 30 SELECT

SELECT se encarga de proyectar o mostrar datos.

- El nombre de las columnas o campos que estamos consultando puede ser cambiado utilizando AS después del nombre del campo y poniendo el nuevo que queremos tener:

```
SELECT titulo AS encabezado
FROM posts;
```

Existe una función de SELECT para poder contar la cantidad de registros. Esa información (un número) será el resultado del query:

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

Queries de la clase

```
SELECT titulo, fecha_publicacion, estatus
FROM posts
```

Usando AS para asignar un alias

```
SELECT titulo, fecha_publicacion AS publicado_en, estatus AS estado  
FROM posts;
```

Contar los registros

```
SELECT COUNT(*) AS numero_post  
FROM posts;
```

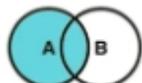
Clase 31 FROM

FROM indica de dónde se deben traer los datos y puede ayudar a hacer sentencias y filtros complejos cuando se quieren unir tablas. La sentencia compañera que nos ayuda con este proceso es **JOIN**.

Los diagramas de Venn son círculos que se tocan en algún punto para ver dónde está la intersección de conjuntos. Ayudan mucho para poder formular la sentencia **JOIN** de la manera adecuada dependiendo del query que se quiere hacer.

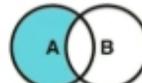
JOIN - Teoría de Conjuntos

Diferencia:

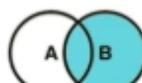


LEFT JOIN
LEFT JOIN

LEFT Join (Con la intersección): Trae todo los datos de la tabla A, que estén o no estén de la tabla B.

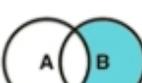


LEFT JOIN (Sin la intersección): Trae todo los datos de la tabla A, que no estén en la tabla B.



RIGHT JOIN
RIGHT JOIN

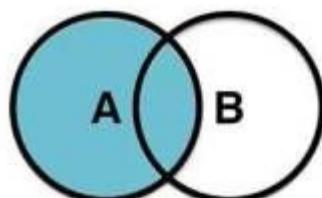
RIGHT JOIN (Con la intersección): Trae todo los datos de la tabla B, que estén o no estén de la tabla A.



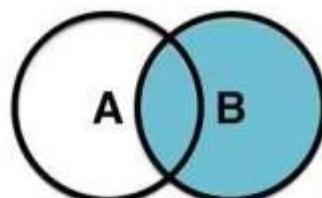
RIGHT JOIN (Sin la intersección): Trae todo los datos de la tabla B, que no estén en la tabla A.

GUIA VISUAL

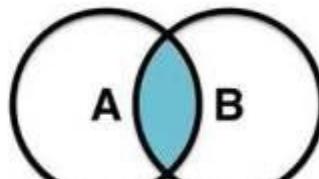
SQL JOINS



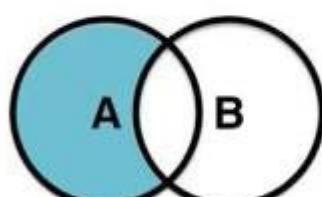
```
SELECT <fields list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



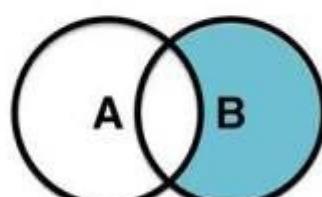
```
SELECT <fields list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



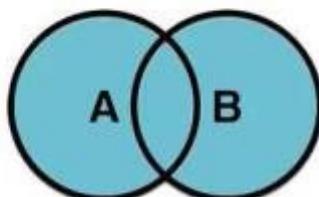
```
SELECT <fields list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



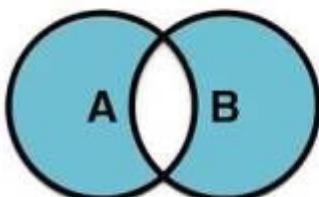
```
SELECT <fields list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <fields list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <fields list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <fields list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL OR B.Key IS NULL
```



Clase 32 Utilizando la sentencia FROM

Comandos de la clase

Left Join trae todos los usuarios (tabla A), tengan o no tengan algún Post (tabla b)

```
SELECT *
FROM usuarios
LEFT JOIN posts ON usuarios.id = posts.usuario_id;
```

Left Join sin intersección

```
SELECT *
FROM usuarios
LEFT JOIN posts ON usuarios.id = posts.usuario_id
WHERE posts.usuario_id IS NULL;
```

Left Join trae todos los Post (tabla B), tengan o no tengan algún Usuario (tabla A)

```
SELECT *
FROM usuarios
RIGHT JOIN posts ON usuarios.id = posts.usuario_id;
```

Right Join sin intersección

```
SELECT *
FROM usuarios
RIGHT JOIN posts ON usuarios.id = posts.usuario_id
WHERE posts.usuario_id IS NULL;
```

Inner Join

```
SELECT *
FROM usuarios
INNER JOIN posts ON usuarios.id = posts.usuario_id;
```

Full Join

```
SELECT *
FROM usuarios
LEFT JOIN posts ON usuarios.id = posts.usuario_id
UNION
SELECT *
FROM usuarios
RIGHT JOIN posts ON usuarios.id = posts.usuario_id;
```

Diferencia Asimétrica

```
SELECT *
FROM usuarios
LEFT JOIN posts ON usuarios.id = posts.usuario_id
WHERE posts.usuario_id IS NULL
UNION
SELECT *
FROM usuarios
RIGHT JOIN posts ON usuarios.id = posts.usuario_id
WHERE posts.usuario_id IS NULL;
```

Clase 33 WHERE

WHERE es la sentencia que nos ayuda a filtrar tuplas o registros dependiendo de las características que elegimos.

- La propiedad **LIKE** nos ayuda a traer registros de los cuales conocemos sólo una parte de la información.
- La propiedad **BETWEEN** nos sirve para arrojar registros que estén en el medio de dos. Por ejemplo los registros con id entre 20 y 30

Queries de la clase

Filtrando de forma numérica

```
SELECT *
FROM posts
WHERE id <= 10 ;
```

Filtrando con strings

```
SELECT *
FROM posts
WHERE estatus = 'activo' ;
```

```
SELECT *
FROM posts
WHERE estatus != 'activo' ;
```

Utilizando LIKE % es un wildcard

```
-- todos los post con la palabra
SELECT *
FROM posts
WHERE titulo LIKE '%escandalo%' ;
```

```
-- todos los post que inicien con la palabra
SELECT *
FROM posts
WHERE titulo LIKE 'escandalo%' ;
```

```
-- todos los post que terminen con la palabra
SELECT *
```

```
FROM posts
WHERE titulo LIKE '%roja' ;
```

Usando WHERE con fechas

```
SELECT *
FROM posts
WHERE fecha_publicacion > "2025-01-01";
```

```
SELECT *
FROM posts
WHERE fecha_publicacion < "2025-01-01";
```

Usando BETWEEN para definir un rango

```
SELECT *
FROM posts
WHERE fecha_publicacion BETWEEN "2025-01-01" AND "2025-12-31";
```

```
SELECT *
FROM posts
WHERE id BETWEEN 5 AND 10;
```

Otros filtros con fechas

```
SELECT *
FROM posts
WHERE YEAR(fecha_publicacion) BETWEEN "2023" AND "2024";
```

```
SELECT *
FROM posts
WHERE MONTH(fecha_publicacion) = "04";
```

Clase 34 Utilizando la sentencia WHERE nulo y no nulo

El valor nulo en una tabla generalmente es su valor por defecto cuando nadie le asignó algo diferente. La sintaxis para hacer búsquedas de datos nulos es **IS NULL**. La sintaxis para buscar datos que no son nulos es **IS NOT NULL**

Para posts sin usuario

```
SELECT *
FROM posts
WHERE usuario_id IS NULL;
;
```

Para posts con usuario

```
SELECT *
FROM posts
WHERE usuario_id IS NOT NULL;
;
```

Para posts con usuario y activo, el uso de AND agrega condiciones de tipo filtro como usar excel

```
SELECT *
FROM posts
WHERE usuario_id IS NOT NULL
AND estatus = 'activo'
AND id < 20
AND categoria_id = 2
AND year(fecha_publicacion) = '2025'
;
```

Clase 35 GROUP BY

GROUP BY tiene que ver con agrupación. Indica a la base de datos qué criterios debe tener en cuenta para agrupar.

Group by te permite agrupar estilo pivot tables, e informes.

```
SELECT estatus, COUNT(*) AS post_quantity
FROM posts
GROUP BY estatus
;
```

Ejemplo 2 Agrupando por Año

```
SELECT YEAR(fecha_publicacion) AS post_year, COUNT(*) AS post_quantity
FROM posts
GROUP BY post_year
;
```

Ejemplo 3 Agrupando por Mes

```
SELECT MONTHNAME(fecha_publicacion) AS post_month, COUNT(*) AS post_quantity
FROM posts
GROUP BY post_month
;
```

Ejemplo 4 Agrupando por Mes y estatus

```
SELECT estatus, MONTHNAME(fecha_publicacion) AS post_month, COUNT(*) AS
post_quantity
FROM posts
GROUP BY estatus, post_month
;
```

Clase 36 ORDER BY y HAVING

La sentencia **ORDER BY** tiene que ver con el ordenamiento de los datos dependiendo de los criterios que quieras usar.

- **ASC** sirve para ordenar de forma ascendente.
- **DESC** sirve para ordenar de forma descendente.
- **LIMIT** se usa para limitar la cantidad de resultados que arroja el query.
- **HAVING** tiene una similitud muy grande con **WHERE**, sin embargo el uso de ellos depende del orden.
Cuando se quiere seleccionar tuplas agrupadas únicamente se puede hacer con **HAVING**.

Ejemplo Order by ascendente default

```
SELECT *
FROM posts
ORDER BY fecha_publicacion
;
```

Ejemplo Order by ascendente explicito

```
SELECT *
FROM posts
ORDER BY fecha_publicacion ASC
;
```

Ejemplo Order by descendente

```
SELECT *
FROM posts
ORDER BY fecha_publicacion DESC
;
```

Ordenando con strings se toma el orden alfabetico.

```
SELECT *
FROM posts
ORDER BY titulo ASC
;
```

Ejemplo Order by ascendente explicito y limite de 5 registros

```
SELECT *
FROM posts
ORDER BY fecha_publicacion ASC
LIMIT 5
;
```

HAVING es similar a **WHERE** aunque no muy utilizada, pero es necesaria cuando quieras hacer un filtro con datos agrupados por un ORDER BY, **HAVING** siempre va despues del **GROUP BY**

```
SELECT MONTHNAME(fecha_publicacion) AS post_month, estatus, COUNT(*) AS
post_quantity
FROM posts
GROUP BY estatus, post_month
HAVING post_quantity > 1
ORDER BY post_month
;
```

Clase 37 El interminable agujero de conejo (Nested queries)

Los **Nested queries** significan que dentro de un query podemos hacer otro query. Esto sirve para hacer join de tablas, estando una en memoria. También teniendo un query como condicional del otro.

Este proceso puede ser tan profundo como quieras, teniendo infinitos queries anidados. Se le conoce como un *producto cartesiano* ya que se multiplican todos los registros de una tabla con todos los del nuevo query. Esto provoca que el query sea difícil de procesar por lo pesado que puede resultar, y se considera como no escalable.

Ejemplo 1

```

SELECT new_table_projection.date, COUNT(*) AS posts_count
FROM (
  SELECT DATE(MIN(fecha_publicacion)) AS date, year(fecha_publicacion) AS post_year
  FROM posts
  GROUP BY post_year
) AS new_table_projection
GROUP BY new_table_projection.date
ORDER BY new_table_projection.date

```

Ejemplo 2

```

SELECT *
FROM posts
WHERE fecha_publicacion = (
  SELECT MAX(fecha_publicacion)
  FROM posts
);

```

Clase 38 Como convertir una pregunta en un query SQL

De pregunta a Query

SELECT: Lo que quieres mostrar **FROM:** De dónde voy a tomar los datos (tablas unicas o con joins) **WHERE:** Los filtros de los datos que quieres mostrar **GROUP BY:** Los rubros por los que me interesa agrupar la información **ORDER BY:** El orden en que quiero presentar mi información **HAVING:** Los filtros que quiero que mis datos agrupados tengan

Clase 39 Preguntandole a la base de datos

GROUP_CONCAT toma el resultado del query y lo pone como campo separado por comas.

Criterios.

DISTINCT: Evita duplicidad en los valores. **ORDER BY:** Sirve para decidir el orden de concatenación del campo. : Es el separador a utilizar para separar los valores (por defecto, el separador es una coma ",").

Pregunta 1 Cuantas etiquetas estan ligadas a los blogposts

```

SELECT posts.titulo, COUNT(*) num_etiquetas
FROM posts
  INNER JOIN posts_etiquetas ON posts.id = posts_etiquetas.post_id
  INNER JOIN etiquetas ON etiquetas.id = posts_etiquetas.etiqueta_id
GROUP BY posts.id
ORDER BY num_etiquetas DESC
;

```

Usando GROUP_CONCAT para tener campos separados por comas

```

SELECT posts.titulo, GROUP_CONCAT(nombre_etiqueta)
FROM posts
INNER JOIN posts_etiquetas ON posts.id = posts_etiquetas.post_id
INNER JOIN etiquetas ON etiquetas.id = posts_etiquetas.etiqueta_id
GROUP BY posts.id
;

```

The screenshot shows the MySQL Workbench interface. At the top, there is a toolbar with various icons. Below the toolbar, the SQL editor contains the query shown above. The results are displayed in a 'Result Grid' table.

titulo	GROUP_CONCAT(nombre_etiqueta)
Se presenta el nuevo teléfono móvil en evento	Eventos,Teléfonos Móviles
Tenemos un nuevo auto inteligente	Computación,Automovilismo
Ganador del premio Nobel por trabajo en genética	Avances,Nobel
Los mejores vestidos en la alfombra roja	Estilo,Moda,Eventos,Celebridades
Los paparazzi captan escándalo en cámara	Celebridades
Se mejora la conducción autónoma de vehículos	Avances,Automovilismo,Computación,Robótica
Se descubre nueva partícula del modelo estandar	Física,Nobel,Avances
Químicos descubren nanomaterial	Avances,Química,Nobel
La bolsa cae estrepitosamente	Largo plazo,Brokers,Inversiones,Bolsa de valores
Bienes raíces más baratos que nunca	Inversiones
Se fortalece el peso frente al dólar	Largo plazo
Tenemos ganador de la formula e	Bolsa de valores
Ganan partido frente a visitantes	Inversiones
Equipo veterano da un gran espectáculo	Campeonatos,Automovilismo
Escándalo con el boxeador del momento	Equipos,Campeonatos
Fútbol OS sacude al mundo	Equipos,Campeonatos,Celebridades
U.S. Robotics presenta hallazgo	Teléfonos Móviles,Computación
Cierra campeonato mundial de fútbol de mano...	Avances
Escándalo en el mundo de la moda	Avances,Robótica
Tenemos campeona del mundial de voleibol	Moda,Celebridades,Equipos,Campeonatos,Cele...

Saber que posts no tienen etiquetas

```

SELECT *
FROM etiquetas
LEFT JOIN posts_etiquetas ON etiquetas.id = posts_etiquetas.etiqueta_id
WHERE posts_etiquetas.etiqueta_id IS NULL
;

```

Clase 40 Consultando PlatziBlog

Puedes usar una abreviación para evitar escribir lo mismo cada vez. Ejemplo:

```
FROM categorias AS c
```

Pregunta: Que categoria tiene mas posts escritos

```
SELECT c.nombre_categoria, COUNT(*) AS cant_posts
FROM categorias AS c
INNER JOIN posts AS p ON c.id = p.categoria_id
GROUP BY c.id
ORDER BY cant_posts DESC
LIMIT 1
;
```

Pregunta: Que persona tiene mas posts escritos

```
SELECT u.nickname, COUNT(*) AS cant_posts
FROM usuarios AS u
INNER JOIN posts AS p ON u.id = p.usuario_id
GROUP BY u.id
ORDER BY cant_posts DESC
;
```

```
SELECT u.nickname, COUNT(*) AS cant_posts, GROUP_CONCAT(nombre_categoria)
FROM usuarios AS u
INNER JOIN posts AS p ON u.id = p.usuario_id
INNER JOIN categorias AS c ON c.id = p.categoria_id
GROUP BY u.id
ORDER BY cant_posts DESC
;
```

Que usuarios no han escrito nada

```
SELECT *
FROM usuarios
LEFT JOIN posts ON usuarios.id = posts.usuario_id
WHERE posts.usuario_id IS NULL
;
```

Modulo 4 Introducción a la bases de datos NO relacionales

Clase 41 Que son y cuales son los tipos de bases de datos no relacionales

Respecto a las bases de datos no relacionales, no existe un solo tipo aunque se engloben en una sola categoría.

Tipos de bases de datos no relacionales

- **Clave - valor:** Son ideales para almacenar y extraer datos con una clave única. Manejan los diccionarios de manera excepcional. Ejemplos: **DynamoDB, Cassandra**.
- **Basadas 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. Ejemplos: **MongoDB, Firestore**.
- **Basadas en grafos:** Basadas en teoría de grafos, sirven para entidades que se encuentran interconectadas por múltiples relaciones. Ideales para almacenar relaciones complejas. Ejemplos: **neo4j, TITAN**.
- **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. Ejemplos: **Memcached, Redis**.
- **Optimizadas para búsquedas:** Pueden ser de diversas estructuras, su ventaja radica en que se pueden hacer queries y búsquedas complejas de manera sencilla. Ejemplos: **BigQuery, Elasticsearch**.

Cada base de datos de este tipo soluciona un problema en particular, pero es mala para todo lo demás dada la gran cantidad de datos.

Clase 42 Servicios administrados y jerarquía de datos

Firebase es un servicio de Google donde puedes tercerizar muchos elementos en la nube. **Jerarquía de datos:**

1. **Base de Datos:** Contiene toda la información que se quiere guardar.
2. **Colección:** Es igual a las tablas en las bases de datos relacionales. Son objetos que agrupan (Documentos) la información que se desea guardar.
3. **Documento:** Es la información que se quiere guardar. Se guarda en un formato muy parecido al formato JSON (es un lenguaje que se utiliza para comunicarse con diferentes lenguajes o aplicaciones). Los documentos dentro de ellos contienen datos.

Jerarquía de datos en firestore



Modulo 5 Manejo de modelos de datos en bases de datos no relacionales

Clase 43 Top level collection con Firebase

El modelo de bases de datos no relacionales es un poco más cercano al mundo real en su comportamiento.

- Las top level collections son las colecciones que se tienen de inmediato o entrada en el proyecto.
- **Firebase** es un servicio que tiene múltiples opciones y está pensado principalmente para aplicaciones móviles y web.

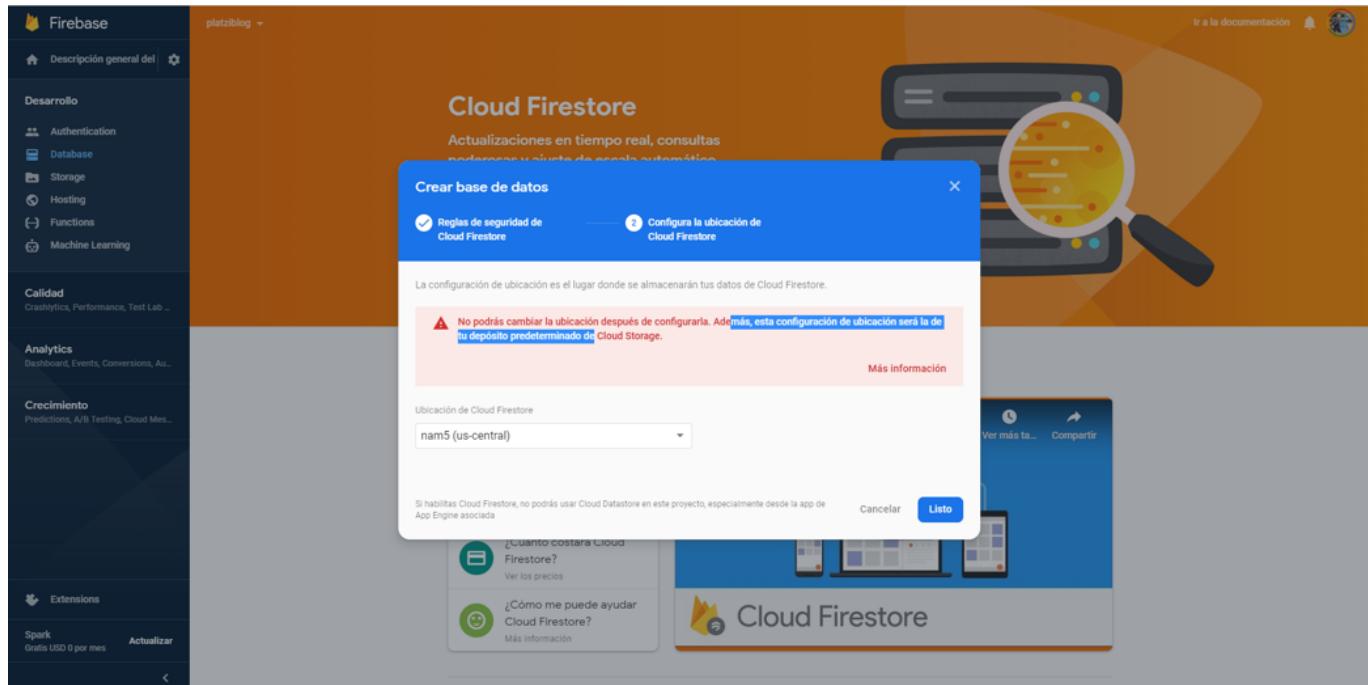
Para esta clase creamos una bd en firebase <https://console.firebaseio.google.com/u/0/?pli=1>

Seguimos los pasos sencillos de la pagina para crear nuestra base de datos "platziblog"

Creamos proyecto

The screenshot shows the Firebase console interface for a project named "platziplata". The main header features the "Cloud Firestore" logo with the tagline "Actualizaciones en tiempo real, consultas poderosas y ajuste de escala automático". A prominent orange button labeled "Crear base de datos" is visible. On the left, a sidebar lists various Firebase services: Desarrollo (Authentication, Database, Storage, Hosting, Functions, Machine Learning), Calidad (Crashlytics, Performance, Test Lab), Analytics (Dashboard, Events, Conversions, Automa...), Crecimiento (Predictions, A/B Testing, Cloud Mess...), Extensions, and Spark (Gratis USD 0 por mes). A "Actualizar" button is located at the bottom of the sidebar.

This screenshot shows the "Crear base de datos" (Create database) dialog for Cloud Firestore. It consists of two main steps: 1. Reglas de seguridad de Cloud Firestore (Cloud Firestore security rules) and 2. Configura la ubicación de Cloud Firestore (Configure Cloud Firestore location). Step 1 contains a note about defining data structure and creating security rules, followed by two radio button options: "Iniciar en modo de producción" (Start in production mode) and "Comenzar en modo de prueba" (Start in test mode). The "Iniciar en modo de producción" option is selected, displaying sample security rules code. Step 2 shows a preview of the database structure with a warning that reads: "Se rechazarán todas las operaciones de lectura y escritura de terceros" (Third-party read and write operations will be rejected). At the bottom, there is a note about Cloud Firestore compatibility with Cloud Datastore and a "Siguiente" (Next) button.



Panel Inicial

	platziplog-aac3f

+ Iniciar colección

Tu base de datos está lista. Solo tienes que agregar datos.

Ubicación de Cloud Firestore: nam5 (us-central)

Para crear una Top level collection (las que estan a nivel de la ruta principal) damos en simbolo plus, y creamos /people

The screenshot shows the Firebase console interface for creating a new collection. The left sidebar includes sections for Authentication, Database, Storage, Hosting, Functions, Machine Learning, Calidad, Analytics, and Crecimiento. The main area displays the 'Database' section for the project 'platziblog'. A modal window titled 'Inicia una colección' (Create Collection) is open, showing step 1: 'Asignar un ID a la colección' (Assign an ID to the collection). It shows the collection path '/people' and a 'Siguiente' (Next) button.

The screenshot shows the continuation of the collection creation process. The modal window is now titled 'Agregar el primer documento' (Add first document). It shows the document path '/people' and a configuration for a new document. The 'ID de documento' (Document ID) field is set to 'ID automático' (Automatic ID). A table defines a single field: 'last_name' of type 'string'. The 'Guardar' (Save) button is visible at the bottom right.

The screenshot shows the Firebase Database interface. On the left, there's a sidebar with various services: Authentication, Database (selected), Storage, Hosting, Functions, Machine Learning, Calidad, Analytics, Crecimiento, Extensions, and Spark. The main area shows a 'Database' view with tabs for Datos, Reglas, Índices, and Uso. Under 'Datos', it shows a hierarchy: people > 3ZLR0MyvYMYH2sbWXM6y. A modal window is open for this specific document, showing fields like 'last_name' with an empty value. At the bottom, it says 'Ubicación de Cloud Firestore: nam5 (us-central)' and the URL 'https://console.firebaseio.google.com/u/0/'.

Clase 44 Creando y borrando documentos en Firestore

Jugamos con la interface observamos los tipos de datos:

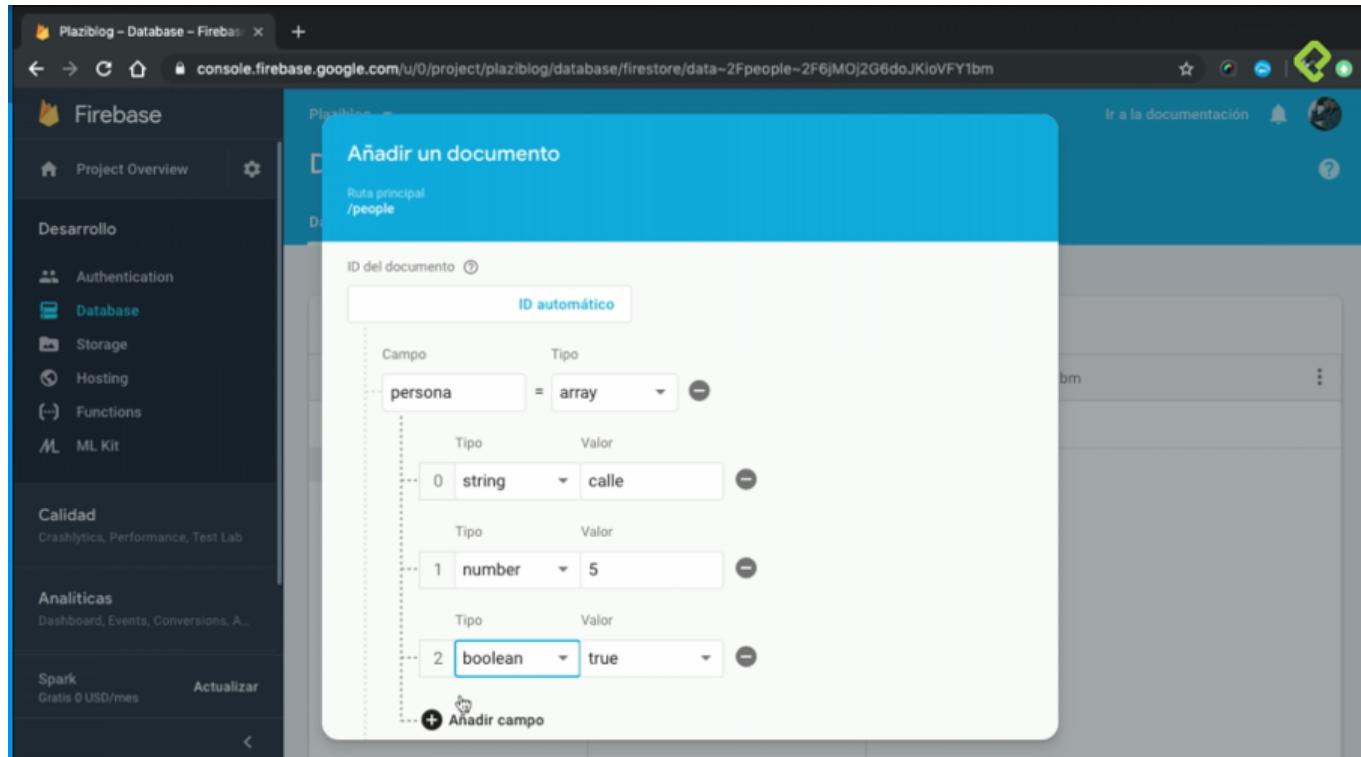
- **string:** (absorbe text, varchar, etc)
- **number:** (int,float)
- **Map:** permite hacer un arreglo dentro de otro arreglo (inception)
- **Timestamp:** para datos de tiempo
- **Geopoint** latitud y longitud.
- **Reference:** hace referencia a otro documento o colección

Ejemplos

The screenshot shows the 'Añadir un documento' (Add document) dialog in the Firebase Database console. The path is set to '/people'. The dialog shows a hierarchical structure of fields:

- Root level: 'persona' (map type)
- Under 'persona': 'direccion' (map type)
 - Under 'direccion': 'calle' (string type) with value 'calle x'
 - Under 'direccion': 'ciudad' (string type) with value 'México'

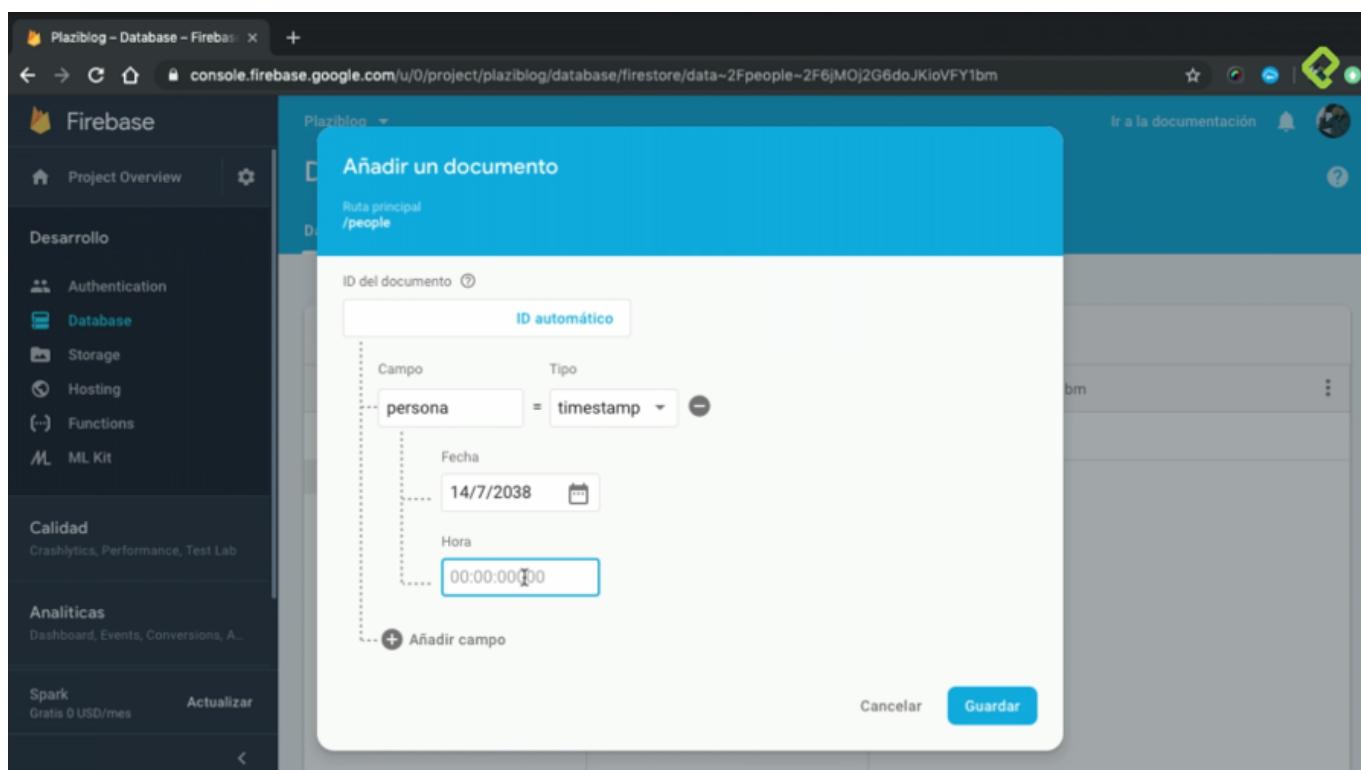
 The 'calle' field under 'direccion' is highlighted with a blue border. The entire dialog is set against a dark background of the Firebase console.



The screenshot shows the Firebase Database console with a document being added under the path `/people`. The document has an ID of `bm`. The structure is defined as follows:

- Campo:** persona **Tipo:** array
- 0:** string **Valor:** calle
- 1:** number **Valor:** 5
- 2:** boolean **Valor:** true

A button at the bottom right of the form says **Añadir campo**.



The screenshot shows the Firebase Database console with a document being added under the path `/people`. The document has an ID of `bm`. The structure is defined as follows:

- Campo:** persona **Tipo:** timestamp

Below the timestamp field, there are two additional fields:

- Fecha:** 14/7/2038
- Hora:** 00:00:00.000

A button at the bottom right of the form says **Añadir campo**. At the very bottom right, there are buttons for **Cancelar** and **Guardar**.

The screenshot shows the Firebase Firestore interface. On the left, there's a sidebar with 'Project Overview', 'Desarrollo' (Development) sections for Authentication, Database, Storage, Hosting, Functions, and ML Kit, and 'Calidad' (Quality) and 'Analíticas' (Analytics) sections. The main area is titled 'Añadir un documento' (Add document) under the path '/people'. It shows a configuration for a new document with an automatically generated ID. A field named 'persona' is defined with a type of 'geopoint'. The 'Latitud' (Latitude) is set to '84' and the 'Longitud' (Longitude) is set to '0'. There are buttons for 'Cancelar' (Cancel) and 'Guardar' (Save).

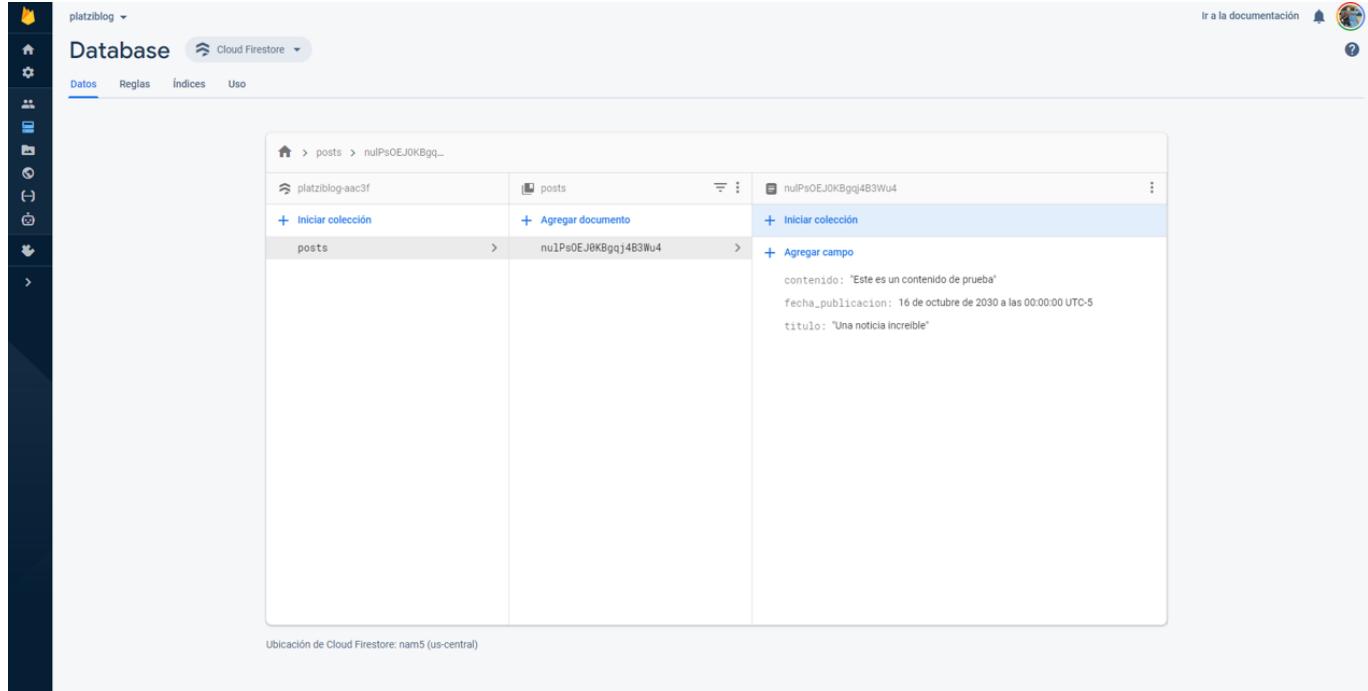
Clase 45 Colecciones vs subcolecciones

La particularidad de las top level collections es que existen en el primer nivel de manera intrínseca. Las subcolecciones ya no vivirán al inicio de la base de datos.

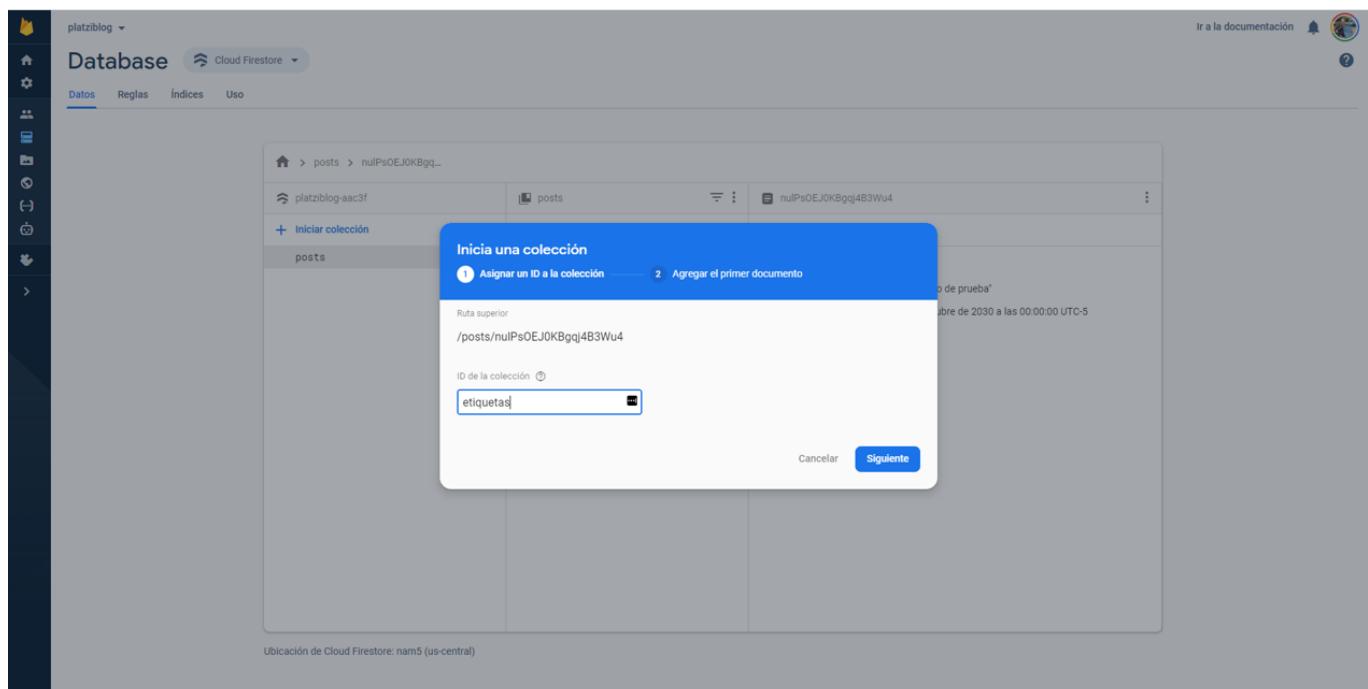
Si tienes una entidad separada que vas a referenciar desde muchos lugares es recomendado usar un top level collection. Por el otro lado si se necesita hacer algo intrínseco al documento es aconsejable usar subcolecciones.

Creamos la colección posts

The screenshot shows the Firebase Database interface. On the left, there's a sidebar with 'Database' and 'Cloud Firestore' tabs, and 'Datos' (Data) selected. The main area is titled 'Inicia una colección' (Start a collection). It shows a step-by-step process: '1 Asignar un ID a la colección' (Assign an ID to the collection) and '2 Agregar el primer documento' (Add the first document). The 'Ruta superior' (Root path) is set to '/' and the 'ID de la colección' (Collection ID) is set to 'posts'. There are buttons for 'Cancelar' (Cancel) and 'Siguiente' (Next).



The screenshot shows the Google Cloud Firestore console interface. On the left is a sidebar with various icons. The main area has a header "Database" and "Cloud Firestore". Below the header are tabs: "Datos" (selected), "Reglas", "Índices", and "Uso". The main content area shows a hierarchical navigation path: "posts > nulPsOEJ0KBgqj4B3Wu4". The right side of the screen displays the contents of the document "nulPsOEJ0KBgqj4B3Wu4", which includes fields: "contenido: \"Este es un contenido de prueba\"", "fecha_publicacion: 16 de octubre de 2030 a las 00:00:00 UTC-5", and "titulo: \"Una noticia increíble\"". At the bottom of the main content area, it says "Ubicación de Cloud Firestore: nam5 (us-central)".



The screenshot shows the Google Cloud Firestore console with a modal dialog titled "Inicia una colección" (Create Collection). The dialog is divided into two steps: 1. Asignar un ID a la colección (Assign an ID to the collection) and 2. Agregar el primer documento (Add the first document). Step 1 is active, showing the collection ID "etiquetas" in a text input field. Step 2 shows a preview of the document structure with fields: "contenido: \"Este es un contenido de prueba\"", "fecha_publicacion: 16 de octubre de 2030 a las 00:00:00 UTC-5", and "titulo: \"Una noticia increíble\"". At the bottom of the dialog are "Cancelar" and "Siguiente" buttons. The background of the main interface is dimmed.

The screenshot shows the Google Cloud Firestore console interface. A modal window titled "Agrega un documento" (Add a document) is open, prompting for document details. The "ID de documento" field is set to "ID automático". Under the "Agregar campo" (Add field) section, a field named "nombre_etiqueta" is defined with type "string" and value "ciencia". The background shows the database structure with collections "posts", "nulPs0EJ0KBgqj4B3Wu4", and "etiquetas". A specific document in the "etiquetas" collection is selected, showing fields like "contenido", "fecha_publicacion", and "titulo". The bottom of the screen displays the URL "https://console.firebaseio.google.com/u/0/project/platziblog-aac3f/overview".

The screenshot shows the Google Cloud Firestore console interface with the database structure visible. The "posts" collection contains a document with ID "nulPs0EJ0KBgqj4B3Wu4", which has fields "contenido", "fecha_publicacion", and "titulo". The "etiquetas" collection contains two documents: one with ID "BzdMMPwSOUvuKR1U2RR" and another with ID "TCxZFUf1qQee1sM16CZ". The bottom of the screen displays the URL "https://console.firebaseio.google.com/u/0/project/platziblog-aac3f/overview".

The screenshot shows the Google Cloud Firestore console interface. On the left is a sidebar with navigation icons. The main area displays a hierarchical database structure under the project 'platziblog'. A specific document path is selected: posts > nulPs0EJ0KBgqj4B3Wu4 > etiquetas > TCxEzFUF1qQee1sMi6CZ. This document contains fields: contenido (containing 'Este es un contenido de prueba'), fecha_publicacion (set to '16 de octubre de 2030 a las 00:00:00 UTC-5'), and titulo (set to 'Una noticia increíble'). To the right of the document, there are options to 'Agregar campo' (Add field) and set its value to 'ciencia'.

Clase 46 Recreando Platziblog

Firestore, es una base de datos basada en documentos, pensada en lo siguiente:

Mantener el estado de tu aplicación. En como se verán reflejados los datos en el frontend para el usuario. Podemos hacer consultas sencillas en base a las top level collection. Ahora si queremos hacer consultas mas complejas podríamos usar big query, que es un data warehouse.

Clase 47 Construyendo Platziblog en Firestore

Construimos categorias "espectaculos","deportes" y "ciencia" como top level collection

The screenshot shows the Firebase Database console. On the left is a sidebar with 'Project Overview', 'Desarrollo' (Authentication, Database, Storage, Hosting, Functions, ML Kit), 'Calidad' (Crashlytics, Performance, Test Lab), and 'Analíticas' (Dashboard, Events, Conversions, A...). The main area shows the 'Database' section with a modal dialog titled 'Iniciar una colección' (Create a collection). The 'Ruta principal del documento' (Main document path) is set to '/categorias'. Under 'ID del documento', the 'ID automático' (Automatic ID) is selected, and a single document entry is shown: 'nombre_categoria' = string, with the value 'Deportes'. At the bottom of the modal are 'Cancelar' (Cancel) and 'Guardar' (Save) buttons. The status bar at the bottom right says 'No has añadido datos.' (You haven't added any data yet).

The screenshot shows the Google Cloud Platform Firestore console. On the left, there's a sidebar with various icons. The main area is titled "Database" and "Cloud Firestore". At the top, there are tabs for "Datos", "Reglas", "Índices", and "Uso". The "Datos" tab is selected. In the center, there's a hierarchical tree view of a database. It starts with a root document "platziBlog-aac3f", which contains a "categorias" collection. This collection has a single document with the ID "mEVHMLjFczz8AdbzP0pk". Inside this document, there's a field named "nombre_categorias" with the value "ciencia". At the bottom of the central pane, it says "Ubicación de Cloud Firestore: nam5 (us-central)".

Creamos en top level la colección usuarios

The screenshot shows the Google Cloud Platform Firestore console. The interface is similar to the previous one, with a sidebar and a "Database" / "Cloud Firestore" header. The "Datos" tab is selected. In the center, a modal window titled "Inicia una colección" (Create a collection) is open. It has two tabs: "Asignar un ID a la colección" (Assign an ID to the collection) and "Agregar el primer documento" (Add the first document). The first tab is selected. The "Ruta superior del documento" (Collection path) is set to "/usuarios". The "ID de documento" (Document ID) is set to "ID automático". Below this, there's a table for defining fields:

Campo	Tipo	Valor
login	string	rusbel
password	string	password_rusbel
nickname	string	Rusbel
email	string	rusbel@platzico

At the bottom of the modal, there are "Cancelar" and "Guardar" buttons.

Creamos la top level posts con dos artículos

Inicia una colección

Ruta superior del documento: /posts

ID de documento: ID automático

Campo	Tipo	Valor
título	string	Se presenta el m...
fecha_publicacion	timestamp	Fecha: 5/4/2030 Hora: 00:00:00000
contenido	string	"Lorem ipsum do"
estatus	boolean	true

Ubicación de Cloud Firestore: nam5 (us-central)

Cancelar Guardar

posts > v3cem9SMmCn...

Agregar documento

Agregar campo

contenido: "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

estatus: true

fecha_publicacion: 5 de abril de 2030 a las 00:00:00 UTC-6

titulo: "Se presenta el móvil en el evento"

Ubicación de Cloud Firestore: nam5 (us-central)

Ahora hacemos la relación del top level collection del usuario /usuarios/qK7F62sQY8bKI8o9w5RY agregando un campo en un documento posts usando el campo reference agregando el usuario

The screenshot shows the Google Cloud Platform Firestore console. A new document is being created under the 'posts' collection. The path is: platziblog-aac3f > posts > v3cem9SMmCn... . The document ID is bGATrSytTmRSmjkuIcG4. A modal window is open for adding a field named 'autor'. The type is set to 'reference'. The value is a document path: /usuarios/qKTF62sQYBk0w5Rf. Below the field, the 'titulo' field is set to "Se presenta el móvil en el evento".

The screenshot shows the Google Cloud Platform Firestore console after the document has been saved. The 'autor' field now contains the value /usuarios/qKTF62sQYBk0w5Rf. The other fields remain the same: 'contenido' with a long Lorem ipsum placeholder, 'estatus' set to true, 'fecha_publicacion' set to '5 de abril de 2030 a las 00:00:00 UTC-6', and 'titulo' set to "Se presenta el móvil en el evento".

Agregamos las referencias para las categorías

Ubicación de Cloud Firestore: nam5 (us-central)

https://console.firebaseio.google.com/u/Q/

Ubicación de Cloud Firestore: nam5 (us-central)

Finalmente agregamos las subcolecciones etiquetas a los posts en agregar colección

The screenshot shows the Google Cloud Firestore interface. At the top, it says 'platziBlog' and 'Database Cloud Firestore'. Below that, there are tabs for 'Datos', 'Reglas', 'Índices', and 'Uso'. The main area shows a hierarchical view of a database. At the top level, there's a collection named 'posts' with a single document 'bGATrSyfTmfnSmjuuWc04'. This document has a subcollection named 'etiquetas' containing a single document '52q4mTWBd6G6zBLyudn'. This document has a field 'nombre_etiqueta' with the value 'telefonos móviles'. Another field 'contenido' contains a large amount of Latin text. There are also other fields like 'autor' and 'categoria'. The bottom of the screenshot shows the URL 'https://console.cloud.google.com/u/0/...

Clase 48 Proyecto final: transformando tu proyecto en una db no relacional

Dentro de las bases de datos relacionales tenemos diferentes niveles de datos. En primer lugar tenemos las Bases de Datos o Esquemas como repositorios donde vivirán los datos que nos interesa guardar. Dentro del esquema existen las Tablas que provienen del concepto de entidades; y a su vez dentro de las tablas tenemos las tuplas o renglones.

Cuando trabajamos con bases de datos basadas en documentos como Firestore, aún existe la figura de la base de datos, sin embargo cambiaremos las tablas en favor de las colecciones y las tuplas en lugar de los documentos.

Recuerda:

Tabla -> Colección

Tupla -> Documento

Dentro de las Colecciones existen 2 grandes tipos. Las Top level collection o colecciones de nivel superior y las subcollections o subcolecciones. Estas últimas viven únicamente dentro de un documento padre.

¿Cómo saber cuál escoger?

Para determinar si tu colección debe ser top level o subcolección no hay una regla escrita en piedra y más bien tiene que ver con el caso de uso en particular y con la experiencia que hayas ganado como desarrollador.

Lo cierto es que no hay una sola forma de estructurar nuestra DB basada en documentos, y por tanto no existe una respuesta correcta, sin embargo a continuación te ofrezco un par de reglas guía que puedes utilizar para transformar tu proyecto que ya trabajaste en bases de datos relacionales en un proyecto no relacional.

Regla 1. Piensa en la vista de tu aplicación

La primera pista que te puedo dar es que pienses en un inicio en la manera en que los datos serán extraídos. En el caso de una aplicación, la mejor forma de pensarlo es en términos de las vistas que vas a mostrar a un

momento determinado en la aplicación.

Es decir, al armar la estructura en la base de datos que sea un espejo o que al menos contenga todos los datos necesarios para llenar las necesidades que tiene nuestra parte visual en la aplicación.

En el caso de Platziblog por ejemplo si tienes una vista de un blog post individual, generalmente conviene mostrar además de los datos inherentes al post como el contenido, datos adicionales como las etiquetas que tiene o por ejemplo el autor (o autores si es colaborativo), en este caso tal vez convenga guardar estas dos "entidades" (autores y etiquetas) como subcolecciones de cada documento blog post.

Regla 2. La colección tiene vida propia

Esta regla se refiere a que la excepción a la regla 1 es cuando tenemos un caso en que la "entidad" que tiene necesidad de vivir y modificarse constantemente de manera independiente a las otras colecciones. Por ejemplo en Platziblog podemos en el ejemplo anterior hacer una excepción a autores porque nos conviene tenerlas como top level collection en el sentido que se añadan, borren, cambien o listen los usuarios sin depender del blog post.

Experimenta aplicando estas dos reglas a un proyecto que ya conozcas en una base de datos relacional y trata de convertirla en un proyecto de Firestore y comentanos los retos a los que te enfrentaste.

Modulo 6 Bases de datos en la vida real

Clase 49 Bases de datos en la vida real

Para datos historicos, queries complejos usar bigquery optimizado para datawarehouse

Para datos del estado de nuestra aplicacion usa colecciones con mongo o firebase.

Puedes usar dos bases de datos en un mismo proyecto, guardar datos de mongo, convertirlos pasarlos a bigquery y hacer analisis sobre ellos.

Clase 50 Big Data

Big Data es un concepto que nace de la necesidad de manejar grandes cantidades de datos. La tendencia comenzó con compañías como **YouTube** al tener la necesidad de guardar y consultar mucha información de manera rápida.

Es un gran movimiento que consiste en el uso de diferentes tipos de bases de datos.

Un ejemplo es cassandra pero tiene ciertos tipos de desventaja, su ventaja manejar datos masivos a gran velocidad

Clase 51 Data warehouse

Data Warehouse trata de guardar cantidades masivas de datos para la posteridad. Allí se guarda todo lo que no está viviendo en la aplicación pero es necesario tenerlo. Debe servir para guardar datos por un largo periodo de tiempo y estos datos se deben poder usar para poder encontrar cuestiones interesantes para el negocio.

Google usa **BigTable**, usa una sola tabla, pero no sirve tanto para hacer consultas

Data Warehouse es un archivo histórico, archivo muerto, en otra base de datos. Sirve para dos actividades principales:

- Guarda una gran cantidad de datos de forma "eterna".
- Poder extraer los datos para hacer analítica.

BigQuery es muy utilizado

Clase 52 Data mining

El **Data Mining** se dedica a minar datos, a extraerlos de donde sea que estén (archivos muertos, base de datos actual, etc...) y hacer sentido de ellos para darles un uso.

Clase 53 ETL

ETL son las siglas de Extract, Transform, Load (extraer, transformar y cargar). Se trata de tomar datos de archivos muertos y convertirlos en algo que sea de utilidad para el negocio. También ayuda a tomar los datos vivos de la aplicación, transformarlos y guardarlos en un data warehouse periódicamente.

Clase 54 Business intelligence

Business Intelligence es una parte muy importante de las carreras de datos ya que es el punto final del manejo de estos. Su razón de ser es tener la información lista, clara y que tenga todos los elementos para tomar decisiones en una empresa. Es necesario tener una buena sensibilidad por entender el negocio, sus necesidades y la información que puede llevar a tomar decisiones en el momento adecuado al momento de realizar business intelligence.

Clase 55 Machine Learning

Machine Learning tiene significados que varían. Es una serie de técnicas que involucran la inteligencia artificial y la detección de patrones. Machine learning para datos tiene un gran campo de acción y es un paso más allá del business intelligence. Nos ayuda a hacer modelos que encuentran patrones fortuitos encontrando correlaciones inesperadas.

Tiene dos casos de uso particulares:

- Clasificación
- Predicción

Clase 56 Data Science

Data Science es aplicar todas las técnicas de procesamiento de datos. En su manera más pura tiene que ver con gente con un background de estadísticas y ciencias duras.

Clase 57 Por qué aprender bases de datos hoy

¡Has concluido el curso! Ahora tienes potentes herramientas y posibilidades para ingresar en este apasionante campo.

Llevaste diagramas a bases de datos, exploraste un poco el mundo de las bases de datos no relacionales, hicimos un proyecto en firestore y transformamos Platzi blog de una base de datos relacional en una base de

datos de documentos.

Dentro de las posibilidades que tienes hoy en día puedes hacer: Machine learning, ETL, Data Warehouse, Data mining, entre otros.

Recuerda practicar mucho con el proyecto. Te invito a que tomes el examen y verifiques tus conocimientos.
¡Éxitos!

Bonus

Clase 58 Bases de datos relacionales vs no relacionales

Las bases de datos relacionales han estado entre nosotros durante un largo tiempo y han sido usadas por grandes como Google, Amazon, entre otros. Aún son usadas por bancos, aseguradoras, etc.

Las bases de datos no relacionales surgen cuando las grandes empresas sintieron necesidad de guardar y extraer grandes cantidades de datos en muy cortos períodos de tiempo, como YouTube.