



TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE TLAXIACO

SEGURIDAD Y VIRTUALIZACIÓN

Nombre de los Integrantes de equipo:

No. Control

Edwin López Santiago

21620123

Yanet González García

21620273

Tema:

Practica 3 : Bases de datos seguras

Docente:

Ing. Osorio Salinas Edward

Carrera:

Ingeniería en Sistemas Computacionales

Grupo: 7US

Semestre: Séptimo.

Tlaxiaco, Oaxaca. A 16 de septiembre de 2024.



Índice

Introducción.....	3
Desarrollo.....	4
Practica 3: Bases de datos seguras.....	4
1. Crea una base de datos en MySQL con una BD que contenga los siguientes campos en tres tablas diferentes:	4
2. Crea un script en Python que permita insertar los datos del archivo `customers-2000000.csv`	9
3. Crea tres usuarios en MySQL con los siguientes permisos:	12
- Usuario 1: Permisos de lectura en la tabla `customers`	12
- Usuario 2: Permisos de lectura y escritura en la tabla `address`	12
- Usuario 3: Permisos de lectura, escritura y eliminación en la tabla `users`	12
4. Crea un script en Python que permita realizar una inyección de SQL en la tabla `users` y que muestre los datos de la tabla `users` en la consola.	13
5. Crea un backup de la base de datos `secure_db` y restaura la base de datos en un servidor diferente.	16
7.- Investiga y describe los conceptos de SQL Injection y cómo se pueden prevenir.	19
8.- Bases de Datos Seguras y cómo se implementan	20
Conclusión.....	23
Bibliografías.....	24



Introducción

La seguridad en bases de datos es un tema fundamental en el desarrollo de aplicaciones, ya que protege la información sensible de los usuarios y garantiza la integridad de los datos. Esta práctica tiene como objetivo aprender a aplicar medidas de seguridad en bases de datos y entender los riesgos que implican vulnerabilidades como la inyección SQL.

A lo largo de esta práctica, se ha creado una base de datos en MySQL con varias tablas, simulando un entorno donde los datos de clientes deben ser protegidos. Esta práctica es importante porque permite entender de manera práctica cómo implementar bases de datos seguras, aplicando diferentes permisos para usuarios y asegurando que solo se acceda a la información necesaria según el rol de cada usuario. A continuación, se presenta la práctica implementada paso por paso y conceptos teóricos para la comprensión del tema.

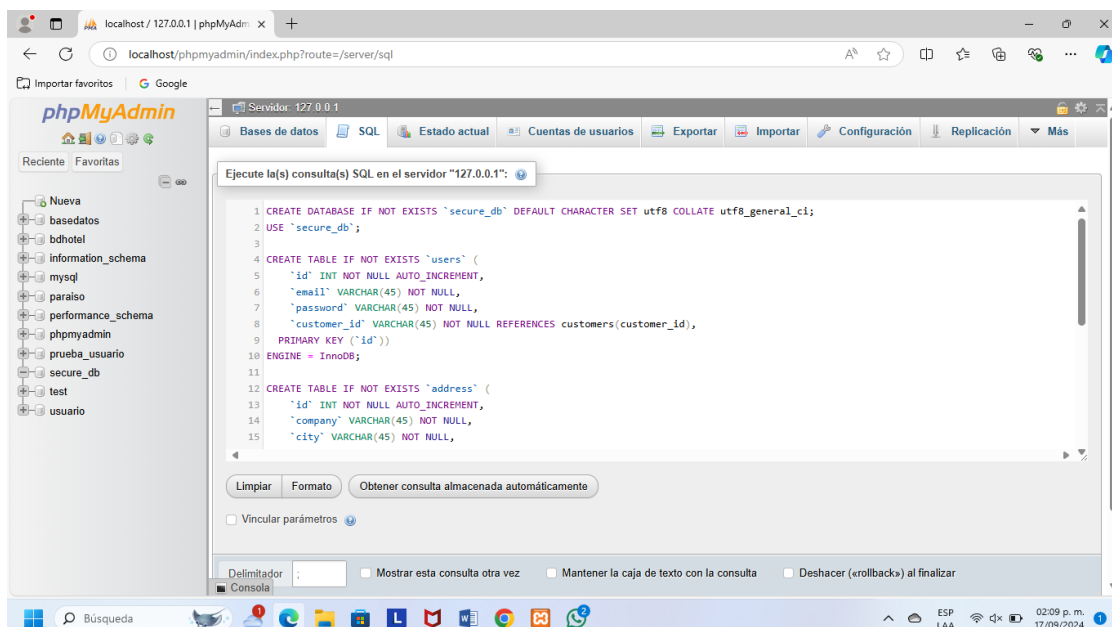
Desarrollo

Practica 3: Bases de datos seguras

1. Crea una base de datos en MySQL con una BD que contenga los siguientes campos en tres tablas diferentes:

```
1. CREATE DATABASE IF NOT EXISTS `secure_db` DEFAULT CHARACTER SET utf8 COLLATE
   utf8_general_ci;
2. USE `secure_db`;
3.
4. CREATE TABLE IF NOT EXISTS `users` (
5.   `id` INT NOT NULL AUTO INCREMENT,
6.   `email` VARCHAR(45) NOT NULL,
7.   `password` VARCHAR(45) NOT NULL,
8.   `customer_id` VARCHAR(45) NOT NULL REFERENCES customers(customer_id),
9.   PRIMARY KEY (`id`))
10. ENGINE = InnoDB;
11.
12. CREATE TABLE IF NOT EXISTS `address` (
13.   `id` INT NOT NULL AUTO_INCREMENT,
14.   `company` VARCHAR(45) NOT NULL,
15.   `city` VARCHAR(45) NOT NULL,
16.   `country` VARCHAR(45) NOT NULL,
17.   `phone_1` VARCHAR(45) NOT NULL,
18.   `phone_2` VARCHAR(45) NOT NULL,
19.   `customer id` VARCHAR(45) NOT NULL REFERENCES customers(customer id),
20.   PRIMARY KEY (`id`))
21. ENGINE = InnoDB;
22.
23. CREATE TABLE IF NOT EXISTS `customers` (
24.   `id` INT NOT NULL AUTO INCREMENT,
25.   `customer_id` VARCHAR(45) NOT NULL,
26.   `first name` VARCHAR(45) NOT NULL,
27.   `last_name` VARCHAR(45) NOT NULL,
28.   `subscription date` DATE NOT NULL,
29.   `website` VARCHAR(45) NOT NULL,
30.   PRIMARY KEY (`id`))
31. ENGINE = InnoDB;
```

Lo primero que vamos a realizar será abrir un gestor de base de datos en phpMyAdmin y en la parte de SQL vamos a pegar nuestro código, este código nos permite crear una base de datos llamada `secure_db` con tres tablas incluidas como lo es para usuarios, `address`, `customers`.





Una vez que ya este pegado nuestro le damos en la opción de continuar y nos marca lo siguiente.

The screenshot shows the phpMyAdmin interface with the 'secure_db' database selected. The 'SQL' tab is active, displaying three successful queries and their results. The first query creates the database 'secure_db'. The second query uses the database. The third query creates a table named 'users' with columns 'id', 'email', 'password', and 'customer_id'. The results show that each query executed successfully without errors.

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

```
USE `secure_db`;
```

```
CREATE TABLE IF NOT EXISTS `users` ( `id` INT NOT NULL AUTO_INCREMENT, `email` VARCHAR(45) NOT NULL, `password` VARCHAR(45) NOT NULL, `customer_id` VARCHAR(45) NOT NULL REFERENCES customers(customer_id), PRIMARY KEY (`id`)) ENGINE = InnoDB;
```

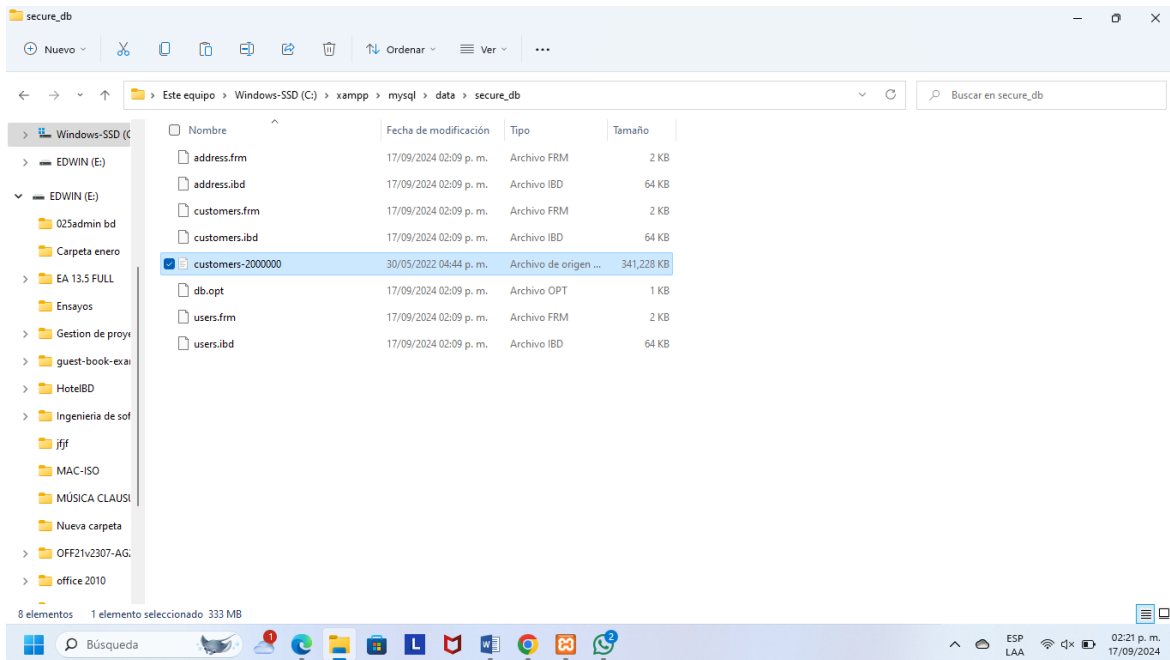
Después de que nos marcara todo correcto y sin ningún error podemos ver nuestras tablas y sus filas de la base de datos

The screenshot shows the phpMyAdmin interface with the 'secure_db' database selected. The 'Estructura' (Structure) tab is active, displaying the list of tables in the database. The tables are 'address', 'customers', and 'users'. The 'users' table is highlighted, showing its structure and the number of rows. The interface also includes a search bar and a 'Crear nueva tabla' (Create new table) button.

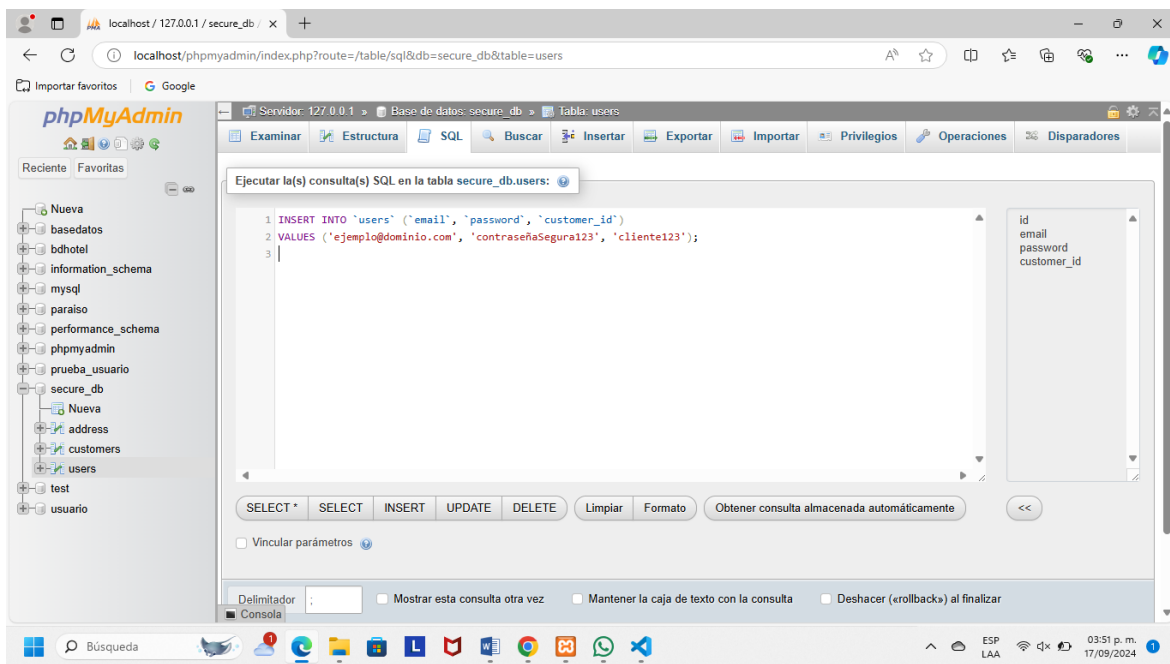
Tabla	Replicación	Acción	Filas	Tipo	Cotejamiento	Tamaño	Residuo a depurar
<input type="checkbox"/> address	✓	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_general_ci	16.0 KB	-
<input type="checkbox"/> customers	✓	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_general_ci	16.0 KB	-
<input type="checkbox"/> users	✓	Examinar Estructura Buscar Insertar Vaciar Eliminar	0	InnoDB	utf8_general_ci	16.0 KB	-
3 tablas			Replicación	Número de filas			
↑			<input type="checkbox"/> Seleccionar todo	Para los elementos que están marcados: ▾			



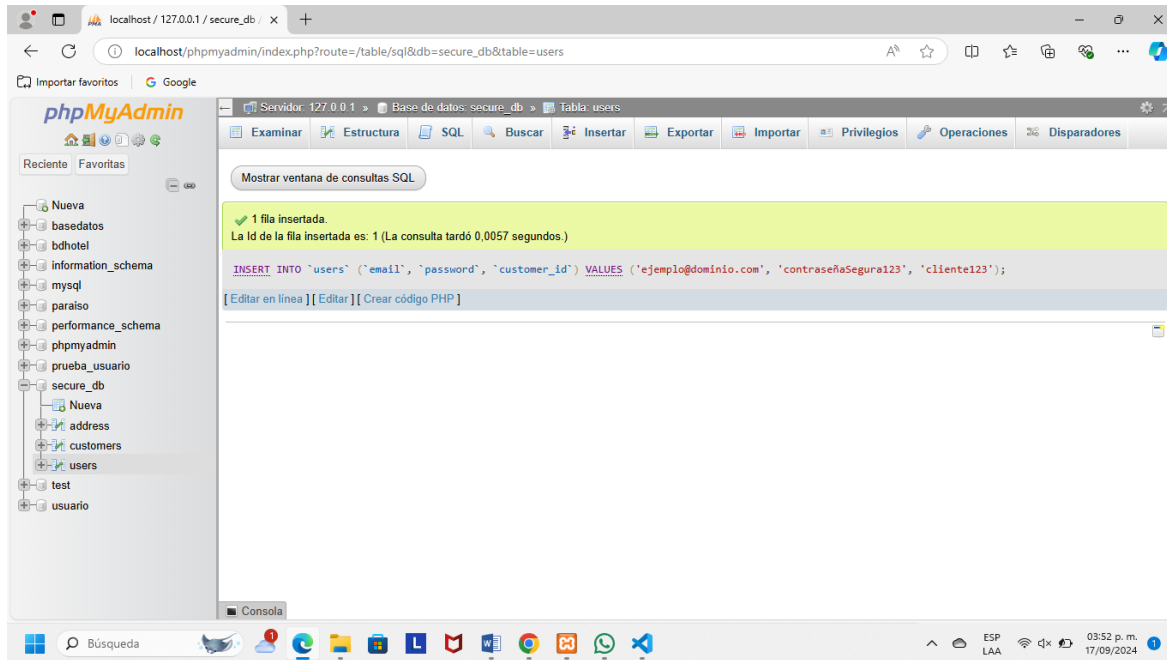
Ahora vamos a pasar en la carpeta donde tenemos los archivos de nuestra base de datos el archivo de customers-2000000, esto lo hacemos para facilitarnos los siguientes pasos.



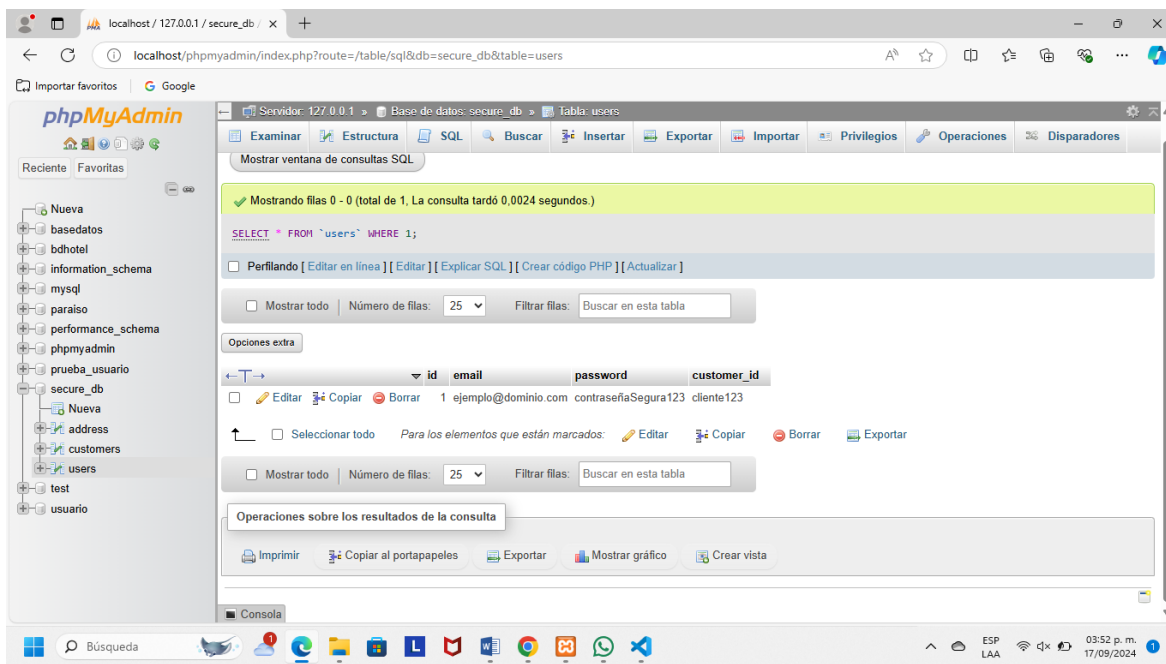
Después vamos a ser una inserción de datos con la siguiente sentencia y datos. Podemos realizar la cantidad que deseamos insertar.



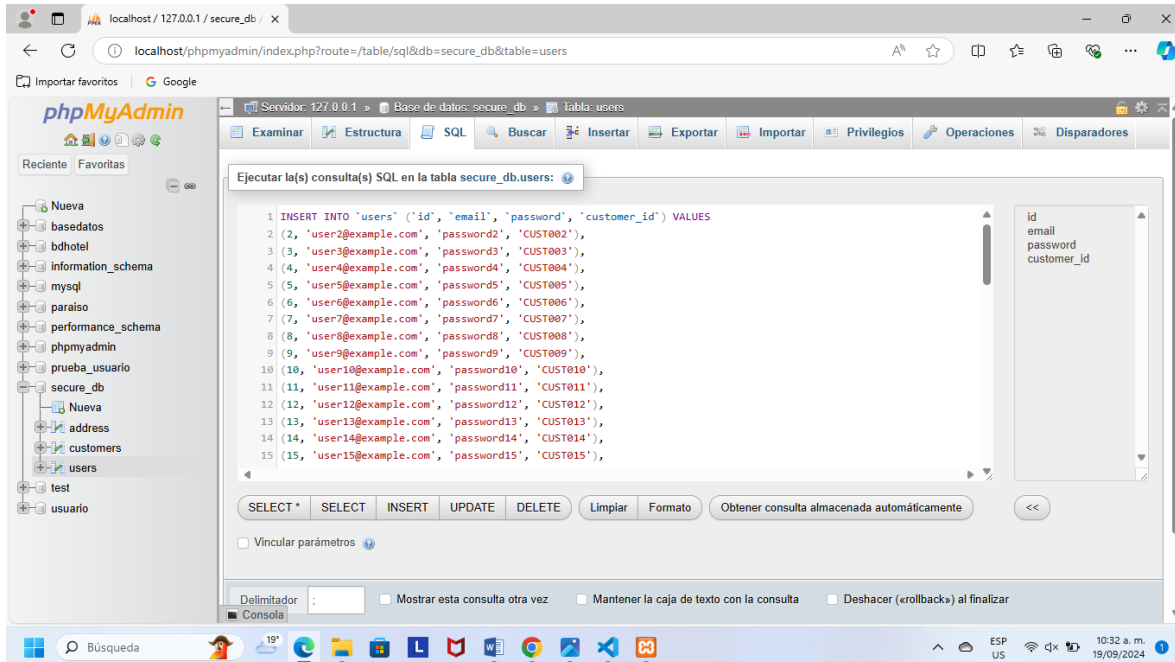
Una vez que tengamos todo listo ya podemos darle en la opción de continuar y nos aparece lo siguiente.



Después podemos ver el dato que insertamos en la tabla de usuarios como se muestra en la imagen. Y así lo podemos hacer con los demás usuarios utilizando la misma sentencia y diferentes datos de los usuarios.



En la siguiente captura podemos ver que estamos haciendo una inserción de Usuarios con sus datos como se ven a continuación.

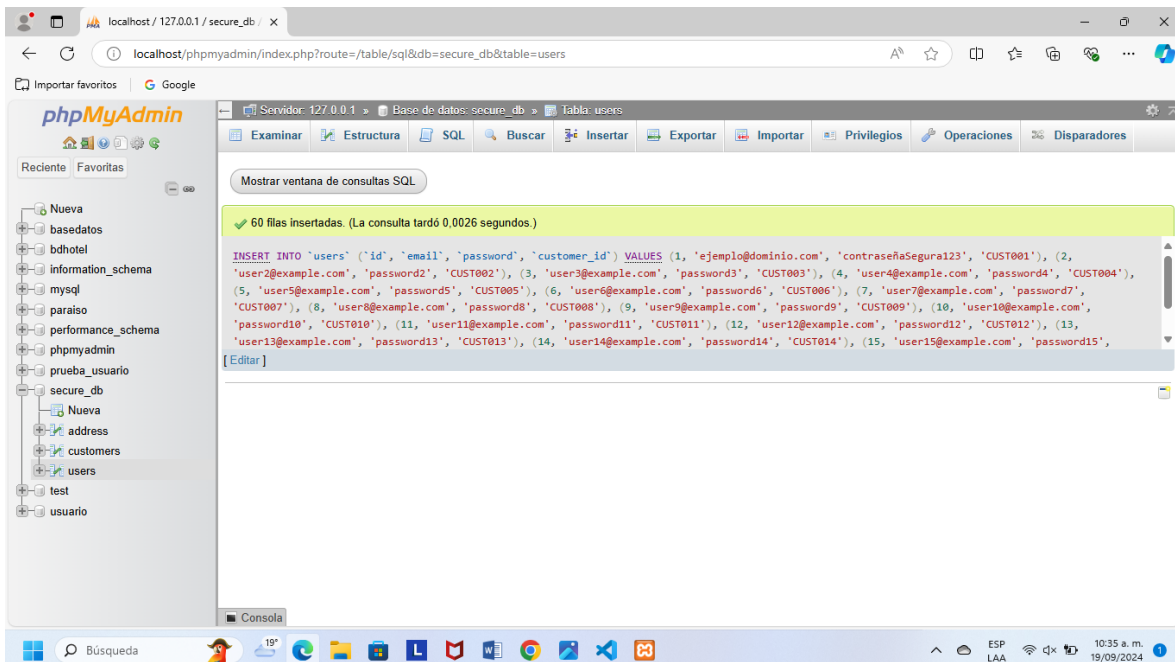


La imagen muestra la interfaz de phpMyAdmin en un navegador web. En la barra de direcciones se ve 'localhost/phpmyadmin/index.php?route=/table/sql&db=secure_db&table=users'. El menú superior incluye 'Examinar', 'Estructura', 'SQL', 'Buscar', 'Insertar', 'Exportar', 'Importar', 'Privilegios', 'Operaciones' y 'Disparadores'. El panel izquierdo muestra el árbol de bases de datos, con 'secure_db' seleccionada y la tabla 'users' visible. El panel central muestra la consulta SQL:

```
1 INSERT INTO `users` (`id`, `email`, `password`, `customer_id`) VALUES
2 (2, 'user2@example.com', 'password2', 'CUST002'),
3 (3, 'user3@example.com', 'password3', 'CUST003'),
4 (4, 'user4@example.com', 'password4', 'CUST004'),
5 (5, 'user5@example.com', 'password5', 'CUST005'),
6 (6, 'user6@example.com', 'password6', 'CUST006'),
7 (7, 'user7@example.com', 'password7', 'CUST007'),
8 (8, 'user8@example.com', 'password8', 'CUST008'),
9 (9, 'user9@example.com', 'password9', 'CUST009'),
10 (10, 'user10@example.com', 'password10', 'CUST010'),
11 (11, 'user11@example.com', 'password11', 'CUST011'),
12 (12, 'user12@example.com', 'password12', 'CUST012'),
13 (13, 'user13@example.com', 'password13', 'CUST013'),
14 (14, 'user14@example.com', 'password14', 'CUST014'),
15 (15, 'user15@example.com', 'password15', 'CUST015');
```

 Debajo de la consulta hay botones para 'SELECT *', 'SELECT', 'INSERT', 'UPDATE', 'DELETE', 'Limpiar', 'Formato' y 'Obtener consulta almacenada automáticamente'. En la parte inferior hay opciones para 'Vincular parámetros', 'Delimitador', 'Mostrar esta consulta otra vez', 'Mantener la caja de texto con la consulta' y 'Deshacer (rollback) al finalizar'.

Le damos en **continuar** y veremos que nuestro registro fue exitoso.



La imagen muestra la misma interfaz de phpMyAdmin, pero ahora se muestra el resultado de la consulta. En la parte superior del panel central, se indica 'Mostrar ventana de consultas SQL'. Debajo, se muestra un mensaje de éxito: '60 filas insertadas. (La consulta tardó 0,0026 segundos)'. A continuación, se repite la consulta SQL, pero ahora incluye 60 registros en lugar de 15. Debajo de la consulta, hay un botón '[Editar]'. El panel izquierdo y la barra superior siguen siendo los mismos.

2. Crea un script en Python que permita insertar los datos del archivo `customers-2000000.csv`

```
1. import pandas as pd
2. import mysql.connector
3. from mysql.connector import Error
4.
5. # Conexión a la base de datos MySQL
6. def create_connection():
7.     try:
8.         connection = mysql.connector.connect(
9.             host='localhost',
10.            database='secure_db',
11.            user='root',
12.            password=''
13.        )
14.        if connection.is_connected():
15.            print("Conexión exitosa a la base de datos")
16.            return connection
17.    except Error as e:
18.        print(f"Error al conectar con MySQL: {e}")
19.        return None
20.
21. def insert_customers_data(connection, customers_df):
22.     cursor = connection.cursor()
23.
24.     insert_query = """
25.     INSERT INTO customers (customer_id, first_name, last_name, subscription_date, website)
26.     VALUES (%s, %s, %s, %s, %s)
27.     """
28.
29.     # Iterar sobre el DataFrame y realizar las inserciones
30.     for index, row in customers_df.iterrows():
31.         try:
32.             cursor.execute(insert_query, (
33.                 row['Customer Id'],
34.                 row['First Name'],
35.                 row['Last Name'],
36.                 row['Subscription Date'],
37.                 row['Website']
38.             ))
39.             connection.commit()
40.         except Error as e:
41.             print(f"Error al insertar datos: {e}")
42.             connection.rollback()
43.
```

```
44. # Leer el archivo CSV y prepararlo para la inserción
45. def read_csv_file(file_path):
46.     try:
47.         # Leer el archivo CSV
48.         customers_df = pd.read_csv(file_path)
49.
50.         # Imprimir las columnas del DataFrame
51.         print("Columnas del archivo CSV:", customers_df.columns.tolist())
52.
53.         # Asegurarse de que las columnas existan y coincidan con la base de datos
54.         print("Archivo CSV leído correctamente")
55.         return customers_df
56.     except Exception as e:
57.         print(f"Error al leer el archivo CSV: {e}")
58.         return None
59.
60. def main():
61.     # Ruta del archivo CSV
62.     file_path = 'customers-2000000.csv'
63.
64.     # Leer el archivo CSV
65.     customers_df = read_csv_file(file_path)
66.
67.     if customers_df is not None:
68.         # Crear la conexión a la base de datos
69.         connection = create_connection()
70.
71.         if connection is not None:
72.             # Insertar datos en la tabla customers
73.             insert_customers_data(connection, customers_df)
74.
75.             # Cerrar la conexión
76.             connection.close()
77.             print("Datos insertados y conexión cerrada correctamente")
78.
79. if __name__ == '__main__':
80.     main()
81.
```

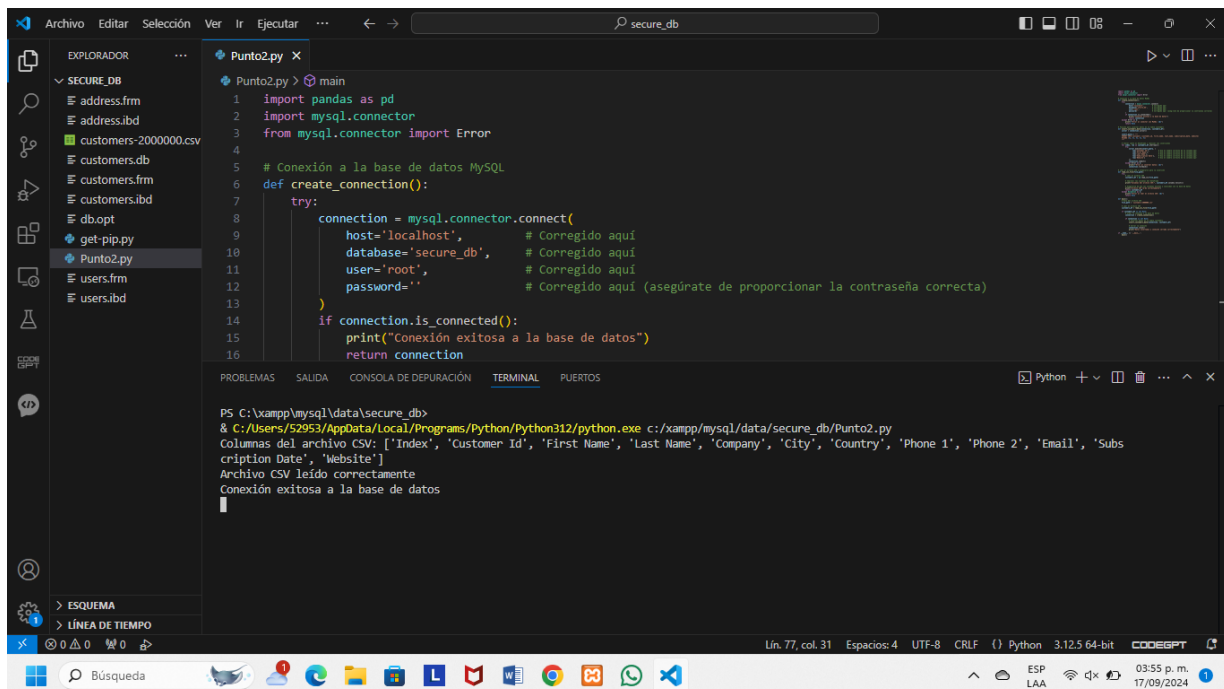
Este código se encarga de **leer un archivo CSV** que contiene datos de clientes y luego **insertar esos datos en una tabla** llamada customers en una base de datos MySQL. El proceso incluye:

1. **Conectar a la base de datos MySQL** usando las credenciales proporcionadas.

2. **Leer el archivo CSV** y cargarlo en un DataFrame de pandas.
3. **Insertar cada fila del CSV en la tabla customers** mediante una consulta SQL de inserción.
4. **Manejo de errores:** Si ocurre un error en cualquier paso, se imprime el mensaje correspondiente y, si es necesario, se revierte la transacción (rollback).
5. **Cerrar la conexión** después de que se hayan insertado los datos.

El código está diseñado para manejar grandes volúmenes de datos, como se puede inferir por el archivo CSV llamado customers-2000000.csv.

Una vez que ya tengamos el Script en python procedemos a ejecutar dicho script y como vemos no nos marcó ningún error.



```
Punto2.py > main
1 import pandas as pd
2 import mysql.connector
3 from mysql.connector import Error
4
5 # Conexión a la base de datos MySQL
6 def create_connection():
7     try:
8         connection = mysql.connector.connect(
9             host='localhost',      # Corregido aquí
10            database='secure_db',   # Corregido aquí
11            user='root',           # Corregido aquí
12            password=''            # Corregido aquí (asegúrate de proporcionar la contraseña correcta)
13        )
14        if connection.is_connected():
15            print("Conexión exitosa a la base de datos")
16        return connection
17
18 # Ejecución principal
19 if __name__ == '__main__':
20     connection = create_connection()
21     if connection is not None:
22         # Leer el archivo CSV
23         df = pd.read_csv('customers-2000000.csv')
24         # Insertar los datos en la tabla customers
25         insert_sql = """
26             INSERT INTO customers (
27                 Index, Customer Id, First Name, Last Name, Company, City, Country, Phone 1, Phone 2, Email, Sub
28                 cription Date, Website
29             ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
30         """
31         cursor = connection.cursor()
32         for index, row in df.iterrows():
33             cursor.execute(insert_sql, row)
34         connection.commit()
35         cursor.close()
36         connection.close()
37         print("Archivo CSV leído correctamente")
38         print("Conexión exitosa a la base de datos")
```

PS C:\xampp\mysql\data\secure_db>
& C:/Users/52953/AppData/Local/Programs/Python/Python312/python.exe c:/xampp/mysql/data/secure_db/Punto2.py
Columnas del archivo CSV: ['Index', 'Customer Id', 'First Name', 'Last Name', 'Company', 'City', 'Country', 'Phone 1', 'Phone 2', 'Email', 'Sub
cription Date', 'Website']
Archivo CSV leído correctamente
Conexión exitosa a la base de datos

La captura de pantalla muestra un script en Python que realiza las siguientes operaciones:

1. **Conectarse a una base de datos MySQL:** Utiliza la función `create_connection()` para establecer conexión con la base de datos local `secure_db` usando el usuario `root` y una contraseña vacía (que debería ser especificada).
2. **Leer datos de un archivo CSV:** La función `read_csv_file()` lee datos desde un archivo CSV nombrado `customers-2000000.csv`, cargándolos en un DataFrame de pandas.
3. **Insertar datos en la base de datos:** La función `insert_customers_data()` toma estos datos y los inserta en la tabla `customers` de la base de datos.

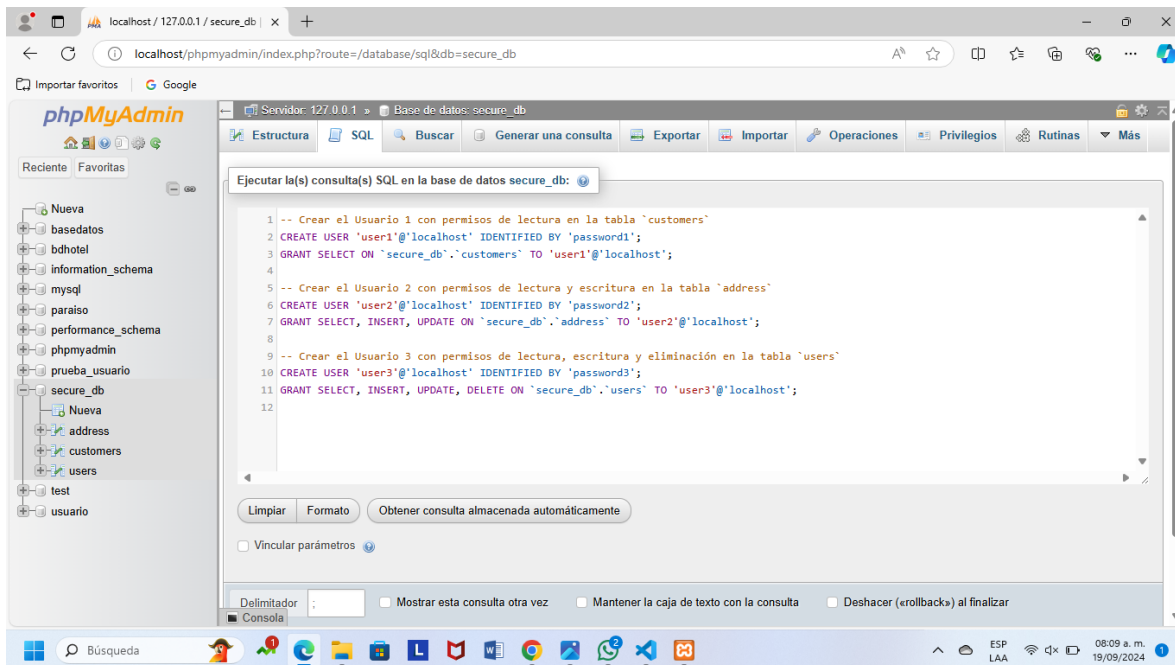
Cada fila del DataFrame se inserta en la base de datos con un manejo de errores para cualquier problema durante la inserción.

4. **Cerrar la conexión:** Una vez completadas las inserciones, el script cierra la conexión a la base de datos.

El terminal confirma que el archivo CSV se ha leído correctamente y que la conexión a la base de datos ha sido exitosa.

3. Crea tres usuarios en MySQL con los siguientes permisos:

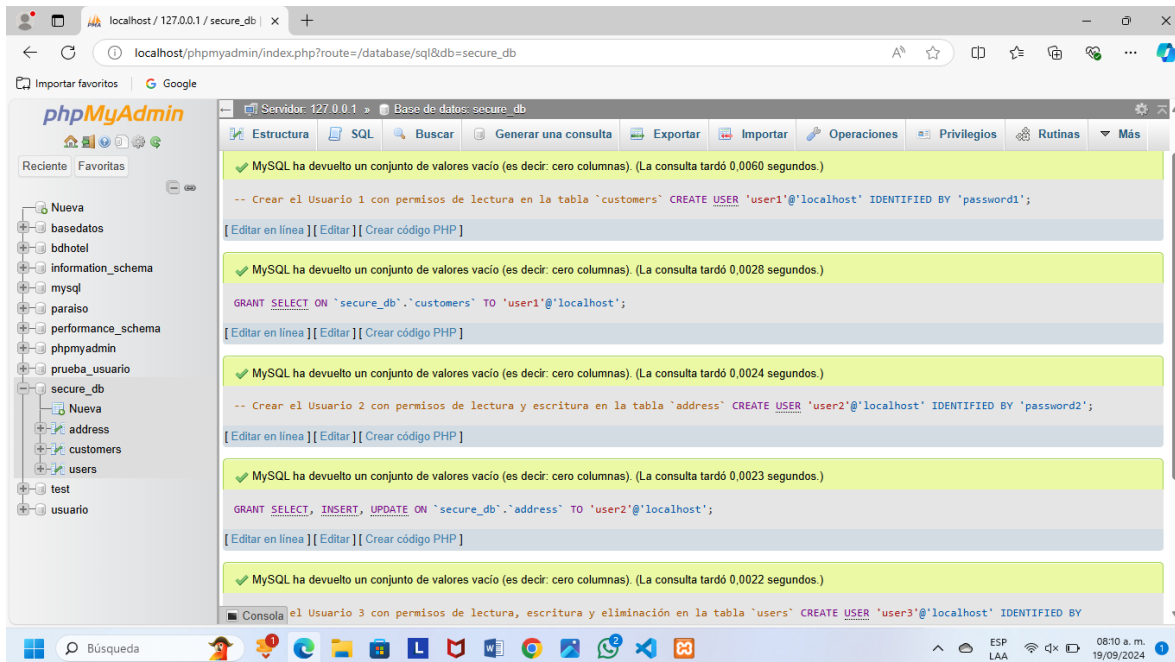
- Usuario 1: Permisos de lectura en la tabla `customers`
- Usuario 2: Permisos de lectura y escritura en la tabla `address`
- Usuario 3: Permisos de lectura, escritura y eliminación en la tabla `users`



Explicación de cada usuario:

- **Usuario 1** (user1): Solo tiene permisos de lectura (SELECT) en la tabla customers.
- **Usuario 2** (user2): Tiene permisos de lectura (SELECT), inserción (INSERT), y actualización (UPDATE) en la tabla address.
- **Usuario 3** (user3): Tiene permisos de lectura (SELECT), inserción (INSERT), actualización (UPDATE), y eliminación (DELETE) en la tabla users.

Después de ingresar el código y darle en continuar veremos que todo está en orden y no nos marca algún error.



4. Crea un script en Python que permita realizar una inyección de SQL en la tabla `users` y que muestre los datos de la tabla `users` en la consola.

```
import mysql.connector

def sql_injection_vulnerable(email_input):
    connection = None
    try:
        # Conectar a la base de datos
        connection = mysql.connector.connect(
            host='localhost',
            user='root',
            password='',
            database='secure_db'
        )

        cursor = connection.cursor()

        # Consulta vulnerable a inyección SQL
        query = f"SELECT * FROM users WHERE email = '{email_input}';"
        print(f"Ejecutando consulta: {query}")
```

```
cursor.execute(query)
result = cursor.fetchall()

# Mostrar resultados de la consulta
for row in result:
    print(row)

except mysql.connector.Error as error:
    print(f"Error: {error}")
finally:
    # Asegurarse de que se cierre la conexión solo si fue exitosa
    if connection and connection.is_connected():
        cursor.close()
        connection.close()

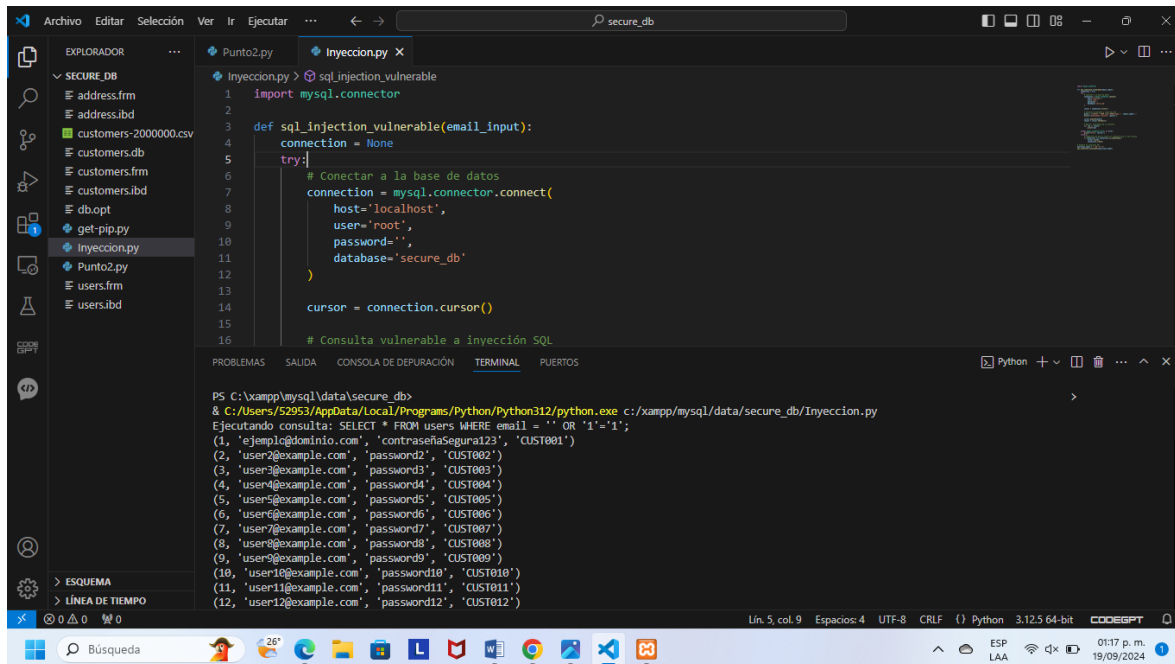
# Ejemplo de inyección SQL
malicious_input = "' OR '1'='1"
sql_injection_vulnerable(malicious_input)
```

Explicación del código de inyección

El código muestra una vulnerabilidad de **inyección SQL** en una aplicación que se conecta a una base de datos MySQL.

1. **Conexión:** Se establece conexión a la base de datos `secure_db`.
2. **Consulta SQL:** La entrada del usuario se inserta directamente en una consulta SQL sin validación, lo que permite a un atacante manipularla.
3. **Ejemplo de inyección:** Con un valor como `"' OR '1'='1"`, la consulta se transforma para devolver todos los registros de la tabla `users`, en lugar de uno específico.
4. **Ejecución y resultados:** Se ejecuta la consulta y se muestran los resultados.
5. **Cierre:** Se cierra la conexión a la base de datos.

Este código es un ejemplo de cómo una mala gestión de la entrada del usuario puede llevar a vulnerabilidades. Para protegerse, es mejor usar consultas parametrizadas.



La imagen muestra una interfaz de desarrollo con un editor de código y una terminal. El editor muestra el archivo `Inyeccion.py` con la siguiente estructura:

```
1 import mysql.connector
2
3 def sql_injection_vulnerable(email_input):
4     connection = None
5     try:
6         # Conectar a la base de datos
7         connection = mysql.connector.connect(
8             host='localhost',
9             user='root',
10            password='',
11            database='secure_db'
12        )
13
14        cursor = connection.cursor()
15
16        # Consulta vulnerable a inyección SQL
```

La terminal muestra la ejecución del script con la siguiente salida:

```
PS C:\xampp\mysql\data\secure_db>
& C:/Users/52953/AppData/Local/Programs/Python/Python312/python.exe c:/xampp/mysql/data/secure_db/Inyeccion.py
Ejecutando consulta: SELECT * FROM users WHERE email = '' OR '1'='1';
(1, 'ejemplo@dominio.com', 'contraseñaSegura123', 'CUST001')
(2, 'user2@example.com', 'password2', 'CUST002')
(3, 'user3@example.com', 'password3', 'CUST003')
(4, 'user4@example.com', 'password4', 'CUST004')
(5, 'user5@example.com', 'password5', 'CUST005')
(6, 'user6@example.com', 'password6', 'CUST006')
(7, 'user7@example.com', 'password7', 'CUST007')
(8, 'user8@example.com', 'password8', 'CUST008')
(9, 'user9@example.com', 'password9', 'CUST009')
(10, 'user10@example.com', 'password10', 'CUST010')
(11, 'user11@example.com', 'password11', 'CUST011')
(12, 'user12@example.com', 'password12', 'CUST012')
```

DESCRIPCIÓN DEL CÓDIGO Y TERMINAL

1) Código Python:

El archivo `Inyeccion.py` contiene la función `sql_injection_vulnerable`.

Se importa la librería `mysql.connector`, que es usada para conectar a la base de datos MySQL desde Python.

Dentro de la función `sql_injection_vulnerable(email_input)`, primero se inicializa la variable `connection` en `None`, y luego, en el bloque `try`, se realiza la conexión a una base de datos llamada `secure_db` utilizando el usuario `root` y una contraseña vacía. Después de establecer la conexión, se obtiene un cursor para ejecutar las consultas SQL.

2) Vulnerabilidad en la consulta SQL:

En la terminal de la parte inferior, se muestra que la consulta SQL que se ejecuta es: `SELECT * FROM users WHERE email = '' OR '1'='1'`;

Esta consulta es vulnerable a inyección SQL. Aquí el campo `email` se pasa como vacío (`''`), y debido al operador `OR '1'='1'`, la condición siempre es verdadera.

Como consecuencia, el servidor de base de datos devuelve todos los registros de la tabla `users`, lo que representa una falla de seguridad.

3) Salida de la consulta en la terminal: La terminal muestra el resultado de ejecutar la consulta SQL anterior, que devuelve todos los registros de la tabla `users`. Algunos ejemplos de las filas devueltas son:

(1, 'ejemplo@dominio.com', 'contraseñaSegura123', 'CUST001')

(2, 'user2@example.com', 'password2', 'CUST002')

(3, 'user3@example.com', 'password3', 'CUST003')

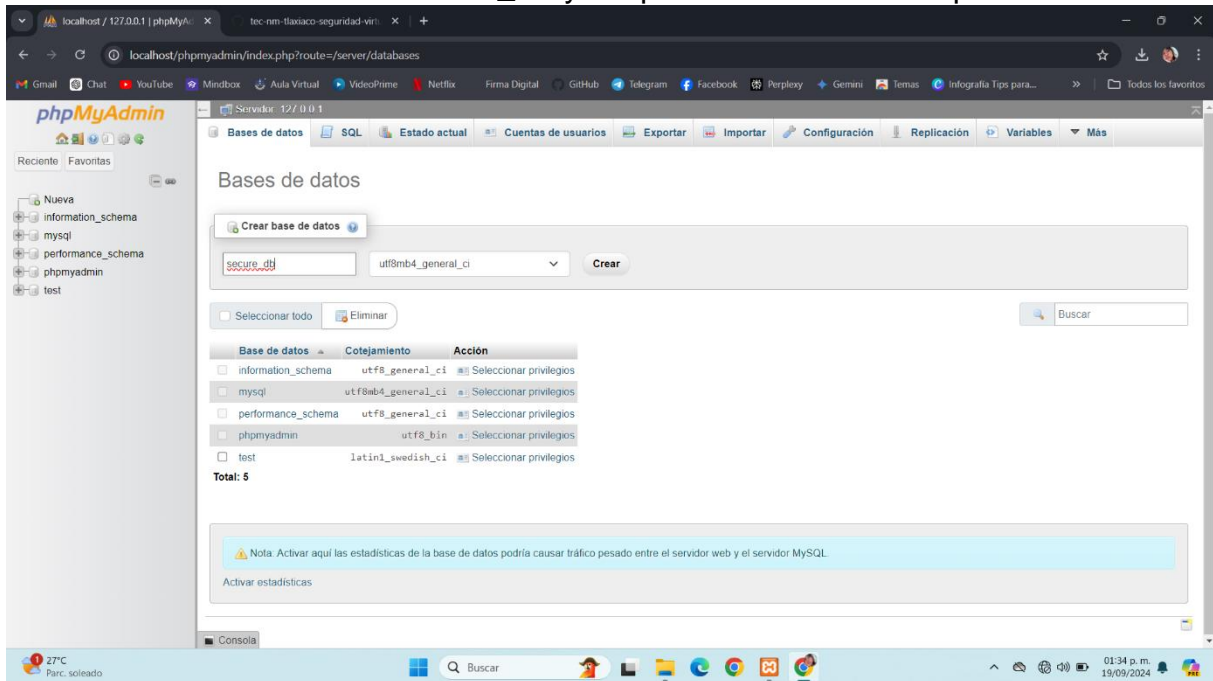
...

(60, 'user60@example.com', 'password60', 'CUST060')

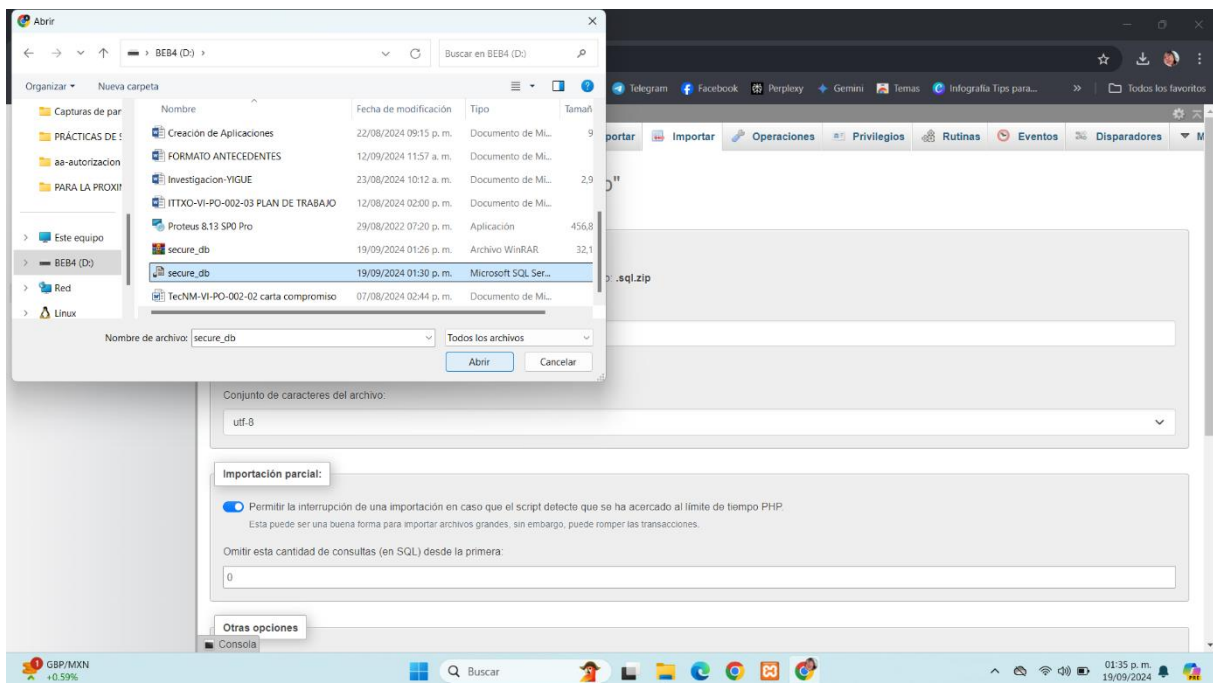
Esto indica que la consulta está recuperando todos los usuarios junto con sus correos electrónicos, contraseñas y un ID de cliente.

5. Crea un backup de la base de datos `secure_db` y restaura la base de datos en un servidor diferente.

Lo primero que vamos a realizar para este procedimiento es crear una nueva base de datos con el nombre de **secure_db** y después le damos en la opción de *crear*.



Después vamos a importar la base de datos y seleccionamos el archivo **secure_bd.sql** y le damos en la opción de *abrir*.





Después que le damos a la opción de **importar** se puede observar que la importación ha sido un éxito y ahora procedemos a verificar las tablas.

La imagen muestra la interfaz de phpMyAdmin en un navegador web. En la parte superior, se indica que la importación se ejecutó exitosamente con 20 consultas ejecutadas. El menú de la izquierda muestra la estructura de la base de datos 'secure_db' con tablas como 'address', 'customers' y 'users'. El panel principal muestra el código SQL de la consulta de importación, incluyendo comandos como 'START TRANSACTION;', 'SET time_zone = '+00:00';' y 'SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT *;'. El estado de la consulta indica que MySQL devolvió un conjunto de valores vacío, lo que sugiere que la importación se completó correctamente sin errores de datos.

Y ahora podemos verificar que las tablas tienen todos los datos a continuación se muestra la tabla de **customers**.

La imagen muestra la interfaz de phpMyAdmin con la tabla 'customers' seleccionada. Se muestra un mensaje de éxito indicando que se están mostrando 11 filas de un total de 1058184. La consulta SQL es 'SELECT * FROM `customers`'. Debajo, se presenta una tabla con los datos de los clientes:

	id	customer_id	first_name	last_name	subscription_date	website
<input type="checkbox"/>	1	4962fdbE6Bfee6D	Pam	Sparks	2020-11-29	https://nelson.com/
<input type="checkbox"/>	2	9b12Ae76fdBc9bE	Gina	Rocha	2021-01-03	https://pineda-rogers.biz/
<input type="checkbox"/>	3	39edfd2F60C85BC	Kristie	Greer	2021-06-20	https://mckinney.com/
<input type="checkbox"/>	4	Fa42AE6a9aD39cE	Arthur	Fields	2020-02-13	https://dominguez.biz/
<input type="checkbox"/>	5	F5702Edae925F1D	Michelle	Blevins	2020-10-20	http://murillo-ryan.com/
<input type="checkbox"/>	6	9FBtbd34c330d9	Greg	Rivers	2020-02-19	https://www.richard.biz/
<input type="checkbox"/>	7	D31D38D576aB1Bf	Brian	Brandt	2020-03-31	https://www.washington.org/
<input type="checkbox"/>	8	3C3B8bee16Bc5F	Dennis	Pacheco	2021-03-20	http://www.armstrong-golden.com/
<input type="checkbox"/>	9	7C1C93522Bd5e4B	Malik	Rivers	2020-01-16	https://watson.com/
<input type="checkbox"/>	10	bABC8cBffe68792	Hayden	Riddle	2022-05-18	https://www.farley.net/
<input type="checkbox"/>	11	534a2B5FE8Aba5d	Edward	Wood	2020-05-22	http://alvarez.com/



Y este es la tabla de datos de los Usuarios.

Mostrando filas 0 - 24 (total de 60, La consulta tardó 0,0004 segundos.)

`SELECT * FROM `users``

☐ Perfilando [[Editar en línea](#)] [[Editar](#)] [[Explicar SQL](#)] [[Crear código PHP](#)] [[Actualizar](#)]

☐ ☐ Mostrar todo | Número de filas: 25 | Filtrar filas: | Ordenar según la clave: Ninguna

Opciones extra

	id	email	password	customer_id
<input type="checkbox"/>	1	ejemplo@dominio.com	contraseñaSegura123	CUST001
<input type="checkbox"/>	2	user2@example.com	password2	CUST002
<input type="checkbox"/>	3	user3@example.com	password3	CUST003
<input type="checkbox"/>	4	user4@example.com	password4	CUST004
<input type="checkbox"/>	5	user5@example.com	password5	CUST005
<input type="checkbox"/>	6	user6@example.com	password6	CUST006
<input type="checkbox"/>	7	user7@example.com	password7	CUST007
<input type="checkbox"/>	8	user8@example.com	password8	CUST008
<input type="checkbox"/>	9	user9@example.com	password9	CUST009
<input type="checkbox"/>	10	user10@example.com	password10	CUST010
<input type="checkbox"/>	11	user11@example.com	password11	CUST011

Hacer un backup de la base de datos `secure_db` es crear una copia de seguridad de los datos para protegerlos de fallos o para moverlos a otro servidor. Restaurar la base de datos en un servidor diferente significa usar ese backup para cargar los datos en el nuevo servidor, asegurando la continuidad del trabajo o distribución de la carga. Esto es útil para migración, recuperación ante desastres, o pruebas.

7.- Investiga y describe los conceptos de SQL Injection y cómo se pueden prevenir.

SQL Injection (Inyección de SQL):

Es una vulnerabilidad de seguridad que permite a un atacante insertar o manipular consultas SQL en una aplicación. El objetivo de una inyección SQL es ejecutar consultas maliciosas que puedan acceder, modificar o eliminar datos en una base de datos. Esta técnica aprovecha las entradas de usuario que no han sido debidamente validadas o sanitizadas antes de ser utilizadas en las consultas SQL.

Tipos comunes de SQL Injection:

- **Inyección SQL clásica:** El atacante inserta directamente código SQL malicioso a través de formularios o URLs.
- **Blind SQL Injection (Inyección SQL ciega):** No se devuelve directamente la información, pero se pueden deducir ciertos datos a través de las respuestas del servidor.
- **Inyección basada en tiempos:** Se basa en la medición de los tiempos de respuesta para determinar si una consulta es verdadera o falsa.

Ejemplo de Inyección SQL:

```
SELECT * FROM users WHERE email = 'email_inyectado' OR '1' = '1';
```

En este ejemplo, la cláusula OR '1' = '1' siempre será verdadera, permitiendo a un atacante obtener acceso no autorizado a la información de la tabla users.

Prevención de SQL Injection:

1. **Usar consultas preparadas (Prepared Statements):** Las consultas preparadas utilizan parámetros en lugar de concatenar las entradas del usuario directamente en la consulta SQL.

```
query = "SELECT * FROM users WHERE email = %s"  
cursor.execute(query, (email,))
```

Esto evita que el código malicioso sea interpretado como parte de la consulta SQL.

2. **Escapar correctamente las entradas:** Las entradas del usuario deben ser adecuadamente escapadas para evitar que se interpreten como parte de la consulta.
3. **Uso de ORM (Object-Relational Mapping):** Utilizar herramientas ORM como SQLAlchemy o Django ORM permite abstraer las consultas SQL y reducir el riesgo de inyecciones SQL.



4. **Validación y sanitización de datos:** Verificar y validar que las entradas del usuario sean del tipo esperado antes de usarlas en la consulta.
5. **Aplicación de políticas de privilegios mínimos:** Asegurarse de que las cuentas de base de datos usadas por las aplicaciones tengan solo los permisos necesarios, limitando así el daño potencial en caso de una inyección SQL.

8.- Bases de Datos Seguras y cómo se implementan

Las bases de datos seguras son aquellas que implementan controles y medidas de seguridad diseñadas para proteger los datos sensibles almacenados frente a accesos no autorizados, alteraciones, robos o ataques. En un entorno donde la información es uno de los activos más valiosos, asegurar las bases de datos es fundamental para proteger la integridad, confidencialidad y disponibilidad de los datos.

Principios de seguridad en bases de datos:

- **Confidencialidad:** Asegura que solo las personas autorizadas puedan acceder a la información almacenada.
- **Integridad:** Garantiza que los datos no se alteren o modifiquen sin autorización.
- **Disponibilidad:** Asegura que los datos estén disponibles cuando se necesiten, evitando caídas o interrupciones prolongadas.

Cómo se implementan las bases de datos seguras:

1. Control de acceso y autenticación robusta:

- **Autenticación:** Se debe verificar la identidad de los usuarios antes de concederles acceso a la base de datos. Los métodos comunes incluyen contraseñas seguras, autenticación de dos factores (2FA) y certificados digitales.
- **Autorización:** Limitar el acceso a los datos con base en los roles y permisos de los usuarios. Implementar el principio de privilegios mínimos, donde cada usuario solo tiene acceso a las partes de la base de datos que necesita.
- **Roles y permisos:** Asignar permisos específicos a diferentes tipos de usuarios (administradores, desarrolladores, usuarios comunes). Esto garantiza que no todos tengan acceso total a la base de datos.

Ejemplo de creación de roles y permisos en MySQL:

```
GRANT SELECT ON secure_db.customers TO 'usuario_lectura'@'localhost'; GRANT SELECT, INSERT, UPDATE ON secure_db.address TO 'usuario_rw'@'localhost'; GRANT ALL PRIVILEGES ON secure_db.users TO 'usuario_admin'@'localhost';
```

2. Cifrado de datos:

- **Cifrado en tránsito:** Los datos que viajan entre la base de datos y los usuarios deben estar cifrados para evitar que sean interceptados por atacantes. Esto se logra mediante el uso de protocolos de seguridad como SSL/TLS.
- **Cifrado en reposo:** Los datos almacenados en la base de datos deben estar cifrados utilizando algoritmos robustos, como AES-256, para que incluso si alguien accede a los discos duros donde se almacena la base de datos, no pueda leer la información.

3. Consultas seguras (evitar inyección SQL): Utilizar **consultas preparadas** o **stored procedures** que sanitizan las entradas de los usuarios antes de utilizarlas en las consultas SQL.

```
query = "SELECT * FROM users WHERE email = %s"  
cursor.execute(query, (email,))
```

Evitar el uso de consultas dinámicas que concatenan entradas de usuarios directamente.

4. Auditoría y monitoreo:

- **Registros de auditoría:** Mantener un registro detallado de todas las actividades que ocurren en la base de datos, incluyendo accesos, modificaciones y eliminaciones. Los logs de auditoría permiten identificar actividades sospechosas y realizar investigaciones post-incidentes.
- **Monitoreo en tiempo real:** Implementar herramientas de monitoreo para detectar patrones de acceso inusuales o comportamientos anómalos en tiempo real. Esto permite respuestas rápidas ante intentos de ataque.

5. Actualización y parches de seguridad:

Los sistemas de bases de datos y sus dependencias deben estar actualizados para protegerse de vulnerabilidades conocidas. Se deben aplicar parches de seguridad y actualizaciones tan pronto como se encuentren disponibles, siguiendo las mejores prácticas de gestión de parches.

6. Backups y recuperación ante desastres:

- **Backups regulares:** Se deben realizar copias de seguridad periódicas de la base de datos para protegerse ante fallos, ataques o desastres.
- **Pruebas de restauración:** Es importante probar regularmente los procedimientos de recuperación para asegurar que los backups sean funcionales y se puedan restaurar rápidamente sin pérdida de datos.



Ejemplo de creación de un backup en MySQL:

```
mysqldump -u usuario -p secure_db > secure_db_backup.sql
```

7. **Segregación de funciones:** Dividir las responsabilidades entre administradores de sistemas, desarrolladores y usuarios finales asegura que ninguna persona tenga un control completo sobre todos los aspectos del sistema, lo que reduce el riesgo de accesos indebidos o abusos internos.
8. **Parcheo y gestión de vulnerabilidades:** Las bases de datos, al igual que otros sistemas de software, pueden tener vulnerabilidades que los atacantes pueden explotar. Las organizaciones deben aplicar regularmente parches de seguridad a sus sistemas de gestión de bases de datos (DBMS) y realizar escaneos de vulnerabilidad.



Conclusión

Realizar esta práctica fue muy útil para comprender mejor la importancia de la seguridad en bases de datos. Durante el proceso, pudimos ver cómo pequeñas vulnerabilidades, como la inyección SQL, pueden comprometer información crítica si no se toman las medidas correctas. También aprendimos a aplicar permisos específicos para diferentes usuarios, lo que es esencial para limitar el acceso y proteger los datos. Esta práctica nos permitió entender de manera práctica cómo proteger las bases de datos y por qué es vital implementar estas medidas de seguridad en el desarrollo de cualquier sistema.

Además, la práctica nos permitió conocer a fondo los riesgos que puede traer una inyección SQL si no se protegen correctamente las consultas en nuestras aplicaciones. Este tipo de ataque es muy común, y nos dimos cuenta de que puede ser explotado fácilmente si no se toman las precauciones adecuadas, como validar correctamente las entradas del usuario y utilizar sentencias preparadas. Esto nos mostró la importancia de escribir un código seguro desde el inicio. Esta práctica nos ayudó a aplicar de forma práctica conceptos clave para la seguridad en bases de datos, y nos enseñó cómo prevenir ataques comunes. Es esencial implementar todas estas medidas en el desarrollo de cualquier sistema para asegurar que los datos de los usuarios estén siempre protegidos. Es así como hemos concluido esta práctica.



Bibliografías

- https://latam.kaspersky.com/resource-center/definitions/sql-injection?srsId=AfmBOorKCYwp9MvEbf4tCU1a8fwRQYJ_bx6XX_8UU2h62Q59XhoiSdpU
- <https://www.cloudflare.com/es-es/learning/security/threats/how-to-prevent-sql-injection/>
- <https://visibilidadweb.unam.mx/capacitacion/perfilesTIC/responsableTIC/Seguridad bases de datos Diplomado ABD.pdf>
- <https://deltaprotect.com/blog/seguridad-para-base-de-datos>
- <https://www.ibm.com/mx-es/topics/database-security>