TECNOLÓGICO
NACIONAL DE MÉXICO®

TECNOLOGICO NACIONAL DE MEXICO

INSTITUTO TECNOLOGICO DE TLAXIACO

REPORTE PRACTICA 3 BASES DE DATOS SEGURAS

YAEL DE JESUS SANTIAGO ORTIZ

JAVIER NOE CRUZ ESPAÑA

KEVIN SANCHES HERNÁNDEZ

PROFESOR: ING. EDWARD OSORIO SALINAS

SEGURIDAD Y VIRTUALIZACION

1U

7° US

ING SISTEMAS COMPUTACIONALES

03 SEP DE 2024

CONTENIDO

OBJETIVO

El objetivo de esta práctica es que el alumno conozca y aplique los conceptos de bases de datos seguras en la seguridad de la información, así como los conceptos de SQL Injection.

INSTRUCCIONES

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

Creamos nuestra base de datos `secure_db` con el siguiente código de SQL en MySQL Workbench con las tablas customers, user y address, debido a que el código que se nos proporcionó nos mostraba errores en sus ejecución se crearon las tablas por aparte para después configurar las claves foráneas:

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

Ilustración 1 Creación de la Base de Datos

```
USE `secure_db`;
CREATE TABLE IF NOT EXISTS `customers` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `customer_id` VARCHAR(45) NOT NULL,
    `first_name` VARCHAR(45) NOT NULL,
    `last_name` VARCHAR(45) NOT NULL,
    `subscription_date` DATE NOT NULL,
    `website` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`id`),
    UNIQUE (`customer_id`) -- Aseguramos que customer_id sea único
)
ENGINE = InnoDB;
```

```sql
CREATE TABLE IF NOT EXISTS `customers` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `customer_id` VARCHAR(45) NOT NULL,
    `first_name` VARCHAR(45) NOT NULL,
    `last_name` VARCHAR(45) NOT NULL,
    `subscription_date` DATE NOT NULL,
    `website` VARCHAR(45) NOT NULL,
    PRIMARY KEY (`id`),
    UNIQUE (`customer_id`) -- Aseguramos que customer_id sea único
)
ENGINE = InnoDB;
```

Ilustración 2 Creación de la tabla `customers`

```sql
CREATE TABLE IF NOT EXISTS `address` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `company` VARCHAR(45) NOT NULL,
    `city` VARCHAR(45) NOT NULL,
    `country` VARCHAR(45) NOT NULL,
    `phone_1` VARCHAR(45) NOT NULL,
    `phone_2` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```

```sql
CREATE TABLE IF NOT EXISTS `address` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `company` VARCHAR(45) NOT NULL,
    `city` VARCHAR(45) NOT NULL,
    `country` VARCHAR(45) NOT NULL,
    `phone_1` VARCHAR(45) NOT NULL,
    `phone_2` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`id`))
ENGINE = InnoDB;
```

Ilustración 3 Creación de la Tabla 'Address'

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

```
26 • CREATE TABLE IF NOT EXISTS `users` (
27       `id` INT NOT NULL AUTO_INCREMENT,
28       `email` VARCHAR(45) NOT NULL,
29       `password` VARCHAR(45) NOT NULL,
30     PRIMARY KEY (`id`))
31   ENGINE = InnoDB;
```

Ilustración 4 Creación de la Tabla 'Users'

Verifica la restricción UNIQUE en customer_id: Asegúrate de que la columna customer_id en la tabla customers tiene una restricción única o es clave primaria. El SQL que proporcionaste inicialmente muestra que agregaste UNIQUE (customer_id). Sin embargo, si MySQL no reconoce esta restricción adecuadamente, debes redefinirla.

Reaplica la restricción UNIQUE si es necesario: Si la columna customer_id no está configurada como única, debes agregar o confirmar la restricción. Utiliza este comando:

```
ALTER TABLE `customers` ADD UNIQUE (`customer_id`);
```

```
ALTER TABLE `customers`
ADD UNIQUE (`customer_id`);
```

Agregar las claves foráneas: Después de confirmar que customer_id es único, vuelve a ejecutar los comandos para agregar las claves foráneas:

```
52 •  ALTER TABLE `address`
53    ADD COLUMN `customer_id` VARCHAR(45) NOT NULL;
54
55 •  ALTER TABLE `address`
56    ADD CONSTRAINT `fk_address_customer`
57    FOREIGN KEY (`customer_id`) REFERENCES `customers` (`customer_id`)
58    ON DELETE CASCADE ON UPDATE CASCADE;
```

```
60 •    ALTER TABLE `users`
61      ADD COLUMN `customer_id` VARCHAR(45) NOT NULL;
62
63 •    ALTER TABLE `users`
64      ADD CONSTRAINT `fk_users_customer`
65      FOREIGN KEY (`customer_id`) REFERENCES `customers` (`customer_id`)
66      ON DELETE CASCADE ON UPDATE CASCADE;
```

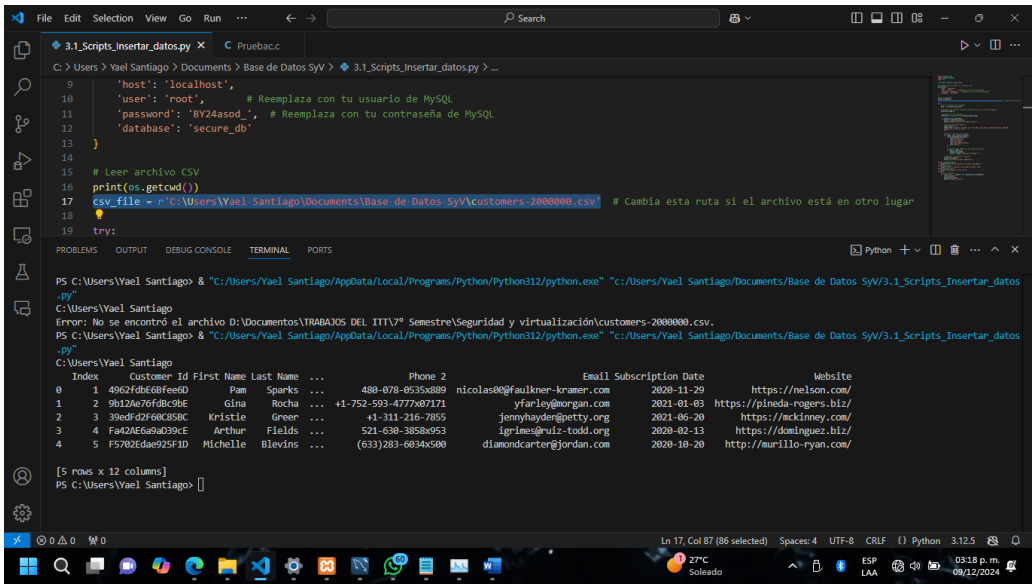
2. Crea un script en Python que permita insertar los datos del archivo `customers-2000000.csv`

```
import pandas as pd
import mysql.connector
from mysql.connector import Error
# Configuración de conexión a la base de datos
db_config = {
    'host': 'localhost',
    'user': 'root',       # Reemplaza con tu usuario de MySQL
    'password': 'BY24asod_',
    'database': 'secure_db'
}
# Leer archivo CSV
csv_file = csv_file = r'C:\Users\Yael Santiago\Documents\Base
de Datos SyV\customers-2000000.csv'# Cambia esta ruta si el
archivo está en otro lugar
try:
    # Leer el archivo CSV con pandas
    data = pd.read_csv(csv_file)
    # Verificar las primeras filas para asegurarse de que se
lee correctamente
    print(data.head())
    # Conectar a la base de datos
    connection = mysql.connector.connect(**db_config)

    if connection.is_connected():
        cursor = connection.cursor()
        print("Conexión exitosa a la base de datos.")
        # SQL para insertar registros
        insert_query = """
        INSERT INTO customers (customer_id, first_name,
last_name, subscription_date, website)
        VALUES (%s, %s, %s, %s, %s)
```

```
        """
        # Insertar cada fila en la tabla
        for index, row in data.iterrows():
            cursor.execute(insert_query, (
                row['customer_id'],
                row['first_name'],
                row['last_name'],
                row['subscription_date'],
                row['website']
            ))
            # Confirmar cada 1000 filas para mayor eficiencia
            if index % 1000 == 0:
                connection.commit()
                print(f"{index} registros insertados.")
        # Confirmar los registros restantes
        connection.commit()
        print("Inserción de datos completada.")
except FileNotFoundError:
    print(f"Error: No se encontró el archivo {csv_file}.")
except Error as e:
    print(f"Error al conectar con la base de datos: {e}")
except Exception as e:
    print(f"Se produjo un error: {e}")
finally:
    # Cerrar conexión
    if 'connection' in locals() and
connection.is_connected():
        cursor.close()
        connection.close()
        print("Conexión cerrada.")
```

Resultados de la ejecución del código y la inserción de los datos solicitados



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`

Crear usuarios:

Cada usuario es creado con un nombre de usuario y una contraseña.

'user1'@'localhost': El nombre del usuario es user1, y solo puede conectarse desde el host localhost.

Creamos el usuario 1 con nombre Javier1

```
1 •  CREATE USER 'javier1'@'localhost' IDENTIFIED BY 'password1';
```

✔	12	15:27:20	CREATE USER 'javier1'@'localhost' IDENTIFIED BY 'password1'	0 row(s) affected

Creamos el usuario 2

```
4 •  CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2';
```

✔	15	15:37:28	CREATE USER 'user2'@'localhost' IDENTIFIED BY 'password2'	0 row(s) affected

Creamos del usuario 3

```
7 •  CREATE USER 'yaelso11'@'localhost' IDENTIFIED BY 'password3';
```

✔	17	15:47:04	CREATE USER 'yaelso11'@'localhost' IDENTIFIED BY 'password3'	0 row(s) affected

Asignar permisos:

GRANT SELECT ON secure_db.customers: Permite al usuario consultar (SELECT) los datos de la tabla customers.

```
2 •  GRANT SELECT ON secure_db.customers TO 'javier1'@'localhost';
```

```
13  15:27:20  GRANT SELECT ON secure_db.customers TO 'javier1'@'localhost'  0 row(s) affected
```

GRANT SELECT, INSERT, UPDATE ON secure_db.address: Permite al usuario consultar (SELECT), insertar (INSERT) y actualizar (UPDATE) datos en la tabla address.

```
5 •  GRANT SELECT, INSERT, UPDATE ON secure_db.address TO 'user2'@'localhost';
```

```
16  15:37:48  GRANT SELECT, INSERT, UPDATE ON secure_db.address TO 'user2'@'localhost'  0 row(s) affected
```

GRANT SELECT, INSERT, UPDATE, DELETE ON secure_db.users: Permite al usuario consultar (SELECT), insertar (INSERT), actualizar (UPDATE) y eliminar (DELETE) datos en la tabla users.

```
8 •  GRANT SELECT, INSERT, UPDATE, DELETE ON secure_db.users TO 'yaelso11'@'localhost';
```

```
18  15:47:09  GRANT SELECT, INSERT, UPDATE, DELETE ON secure_db.users TO 'yaelso11'@'localh...  0 row(s) affected
```

Aplicar cambios:

FLUSH PRIVILEGES: Asegura que los cambios en los privilegios se reflejen inmediatamente.

```
10 •  FLUSH PRIVILEGES;
```

```
19  15:50:04  FLUSH PRIVILEGES  0 row(s) affected
```

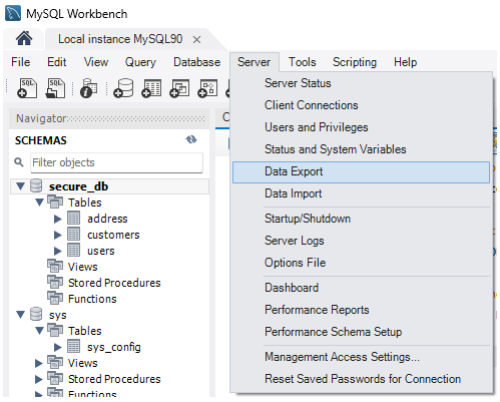
4. Crea un script en Python que permita realizar una inyección de SQL en la tabla `users` y que muestre los datos de la tabla `users` en la consola.

```
import mysql.connector
# Configuración de la base de datos
db_config = {
    'host': 'localhost',
    'user': 'root',  # Cambia por tu usuario de MySQL
    'password': 'BY24asod_',  # Cambia por tu contraseña
    'database': 'secure_db'
```

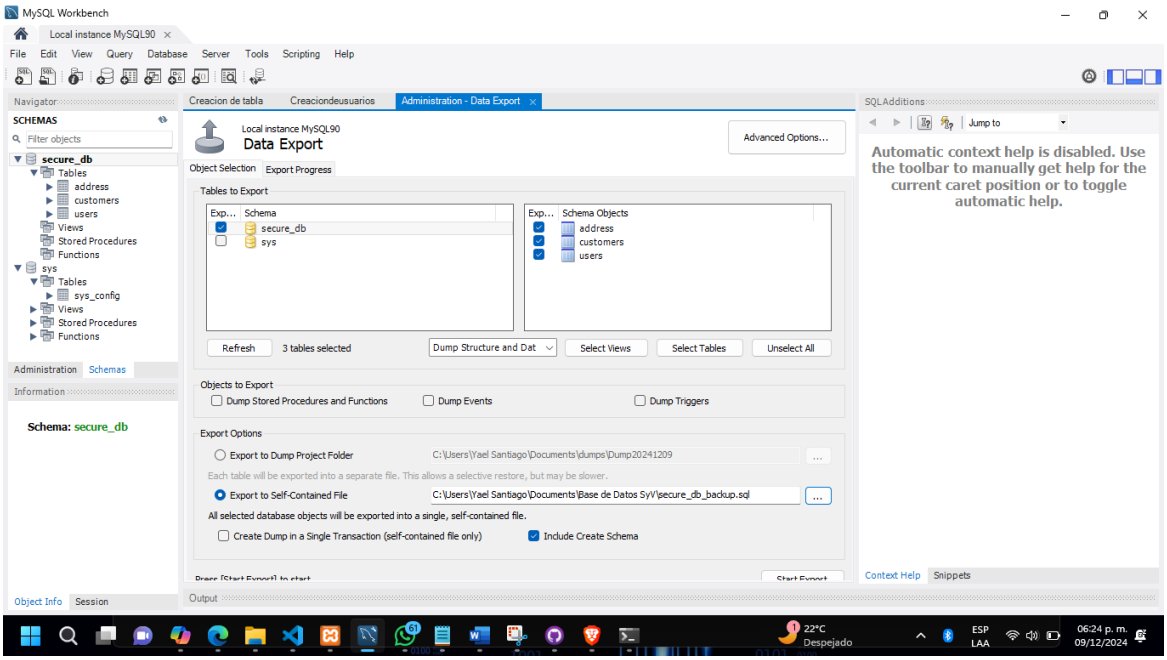
```
}
# Conexión a la base de datos
connection = mysql.connector.connect(**db_config)
cursor = connection.cursor()
# Entrada simulada del usuario
username = input("Introduce tu usuario: ")
password = input("Introduce tu contraseña: ")
# Consulta vulnerable
query = f"SELECT * FROM users WHERE username = '{username}' AND
password = '{password}';"
print(f"Ejecutando consulta: {query}")
cursor.execute(query)
results = cursor.fetchall()
# Mostrar los resultados
if results:
    print("Usuario encontrado:")
    for row in results:
        print(row)
else:
    print("No se encontraron resultados.")
cursor.close()
connection.close()
```

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

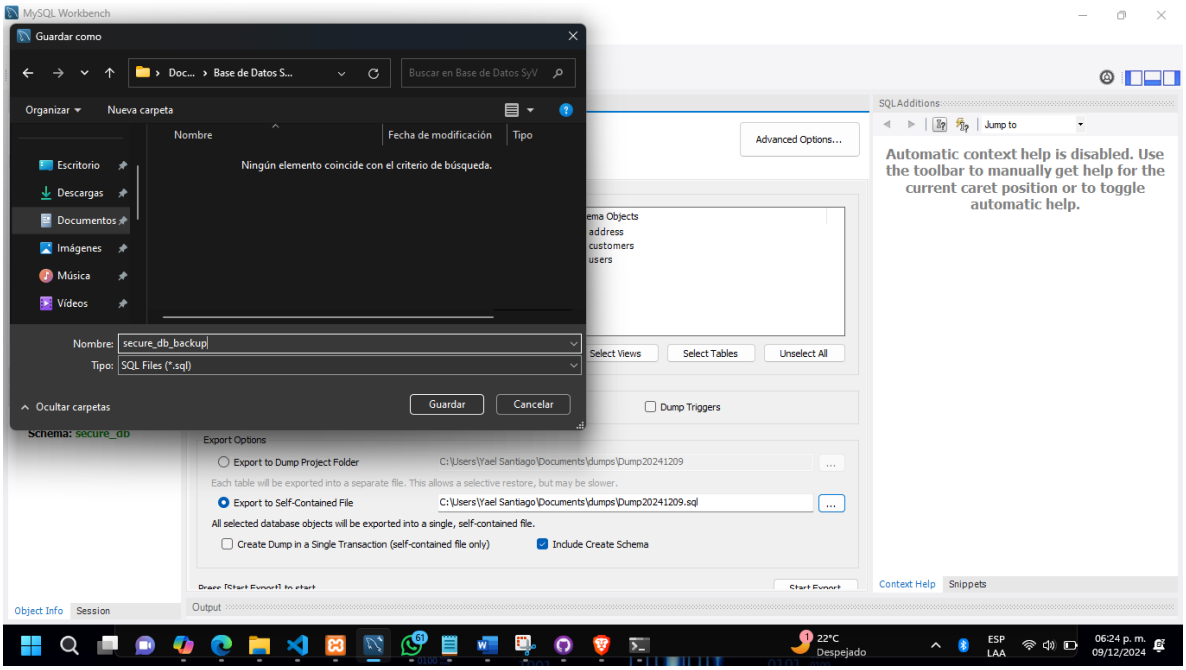
Como la practica la realizamos en MySQL nos facilitó la realización de mismo, primero exportamos nuestra base de datos.



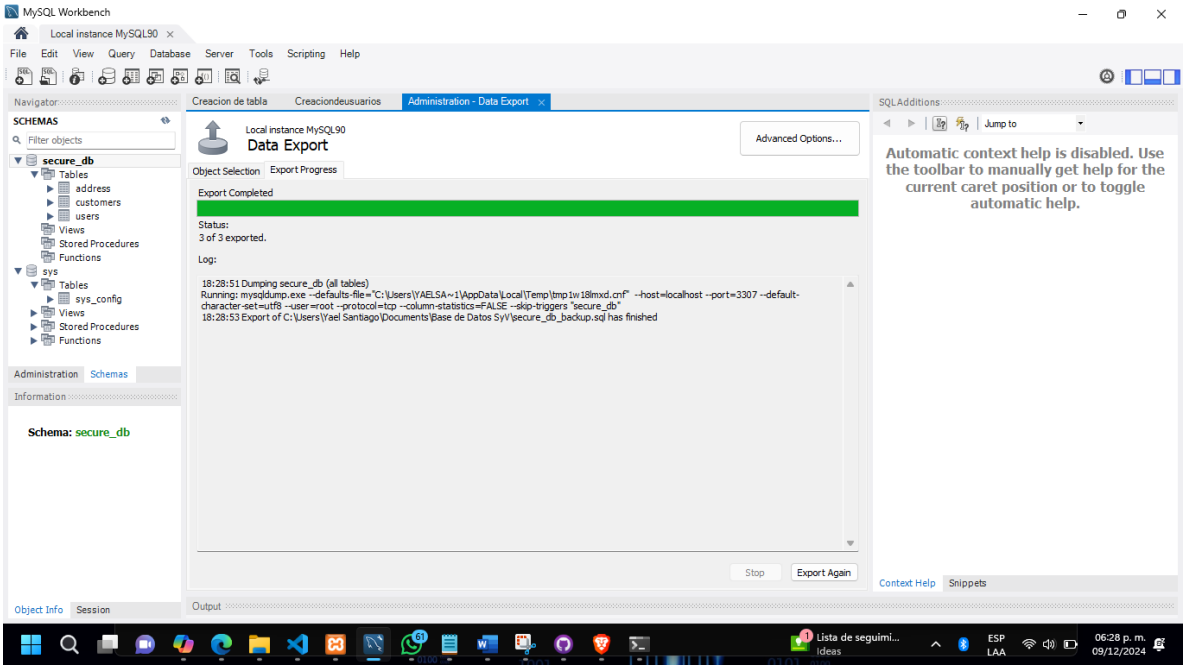
Configuramos la exportación de la base de datos para la posterior implementación en otro servidor.



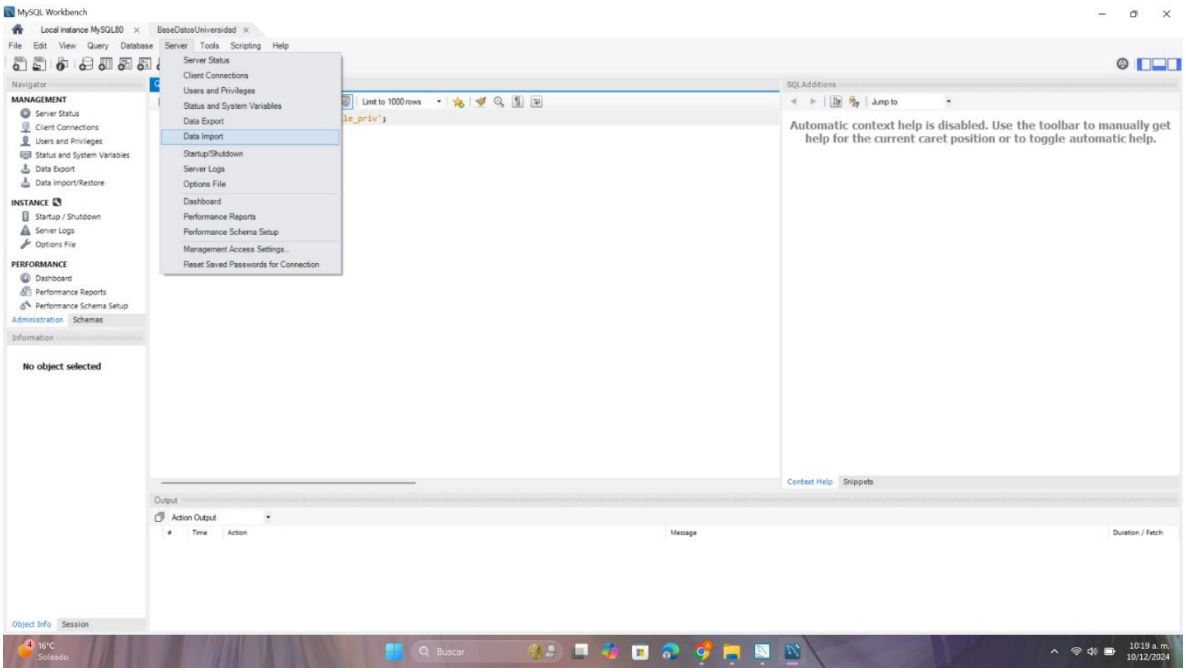
Seleccionamos donde queremos guardar la Base de Datos

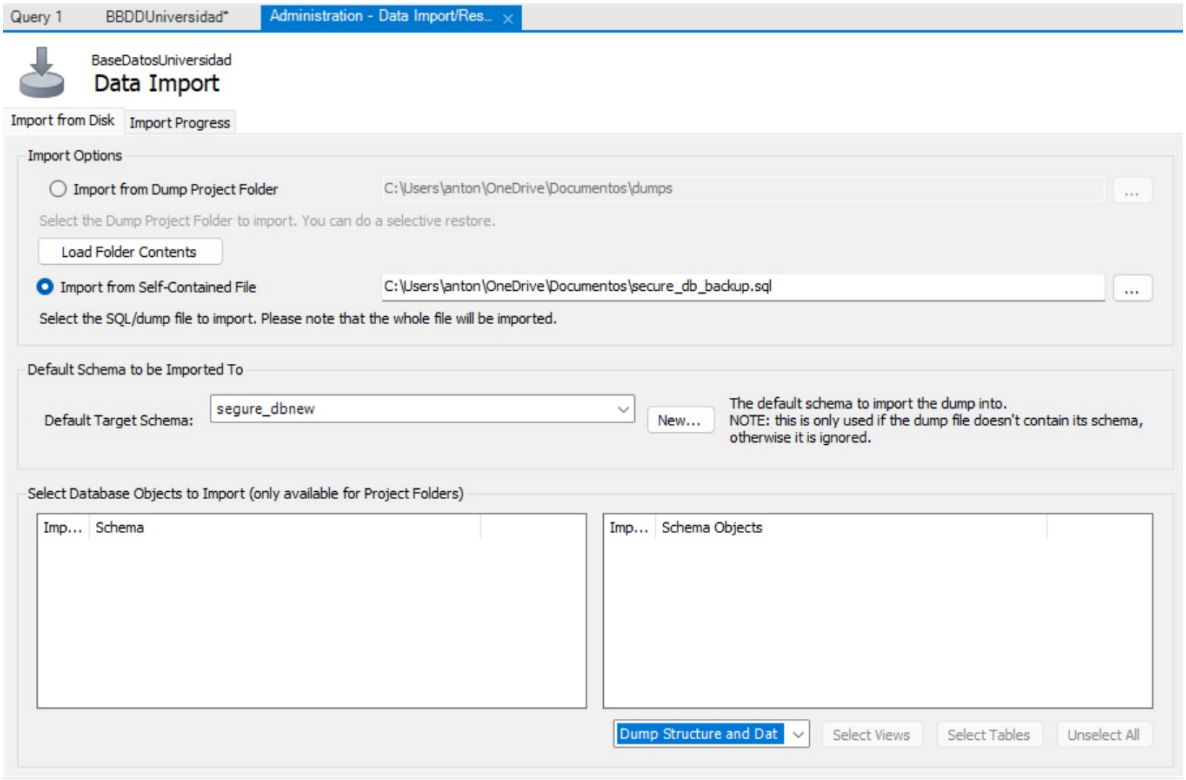


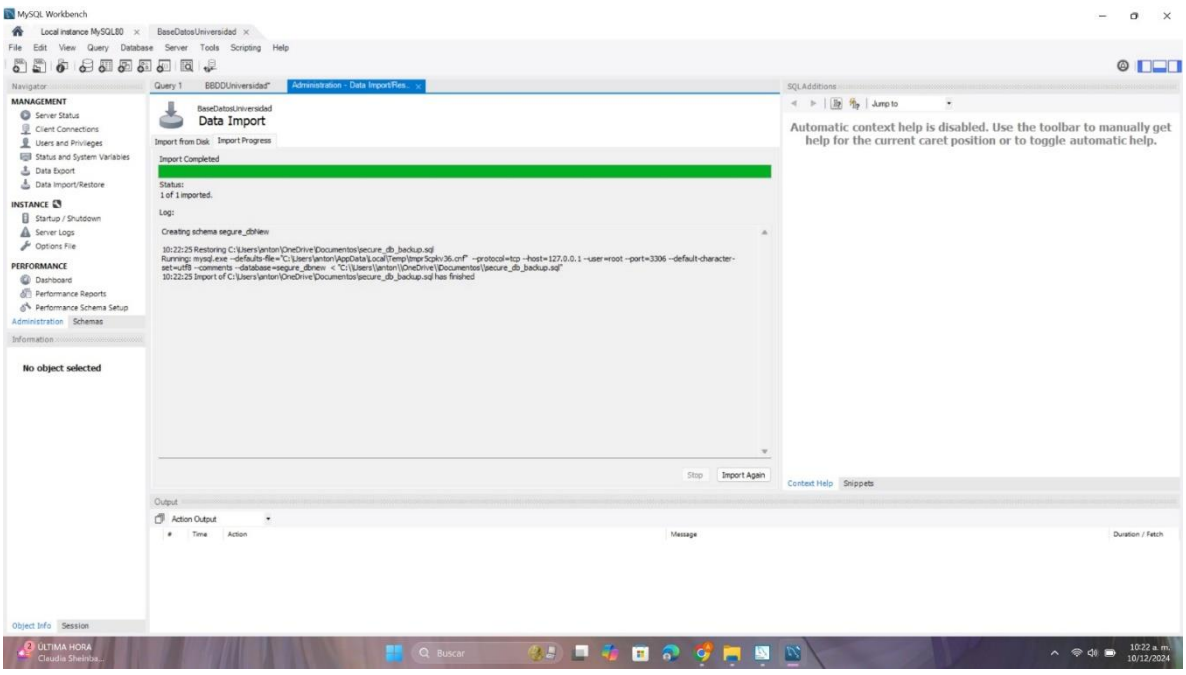
Exportamos la base de datos



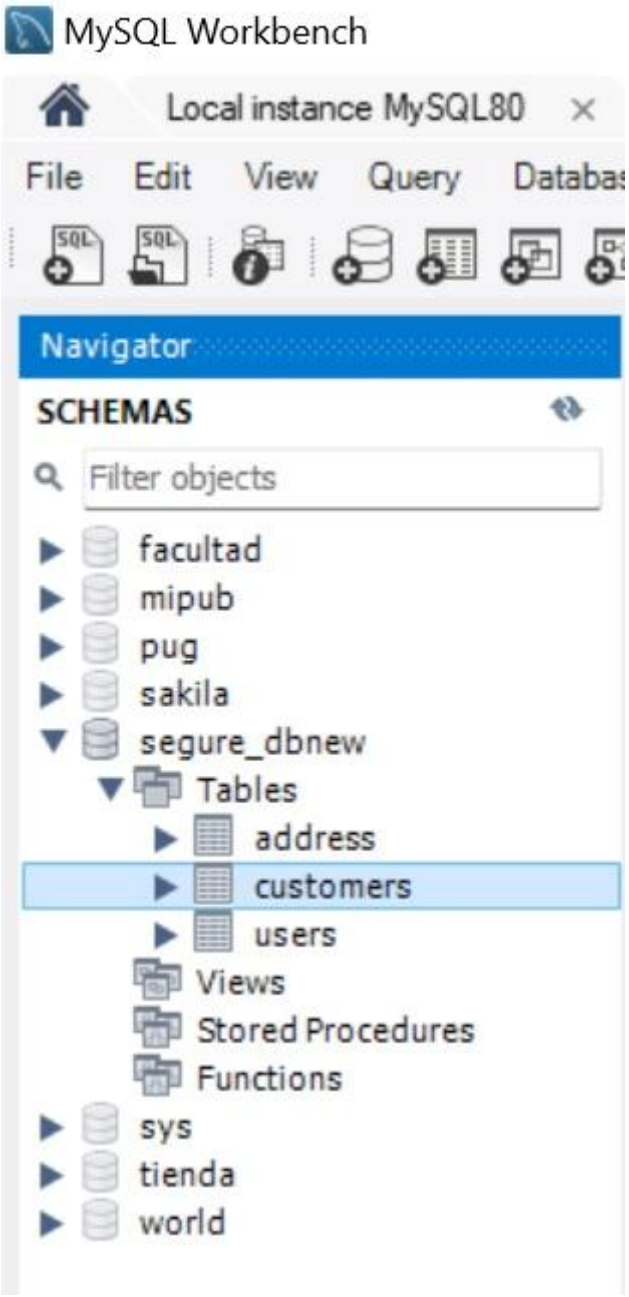
Para importar en otro equipo con MySQL







Verificamos la existencia de la nueva base de datos



INVESTIGACIÓN

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

SQL Injection es una técnica de ataque cibernético en la que un atacante manipula consultas SQL a través de entradas no validadas en una aplicación, obteniendo acceso no autorizado, manipulando o eliminando datos de la base de datos. Este

tipo de ataque ocurre cuando la entrada del usuario se inserta directamente en las consultas SQL sin validarse ni sanearse adecuadamente.

¿Cómo funciona un ataque de SQL Injection?

Manipulación de consultas: Un atacante introduce código SQL malicioso en campos de entrada de usuario.

Ejecución del ataque: Si la aplicación no valida la entrada, el código malicioso se ejecuta junto con la consulta legítima.

Resultados posibles:

- Robo de datos sensibles.
- Modificación o eliminación de registros.
- Acceso total a la base de datos.

Cómo prevenir SQL Injection:

- Consultas parametrizadas (Prepared Statements):
- Utiliza parámetros en lugar de concatenar cadenas.

Validación y sanitización de entradas:

- Restringe el formato y el contenido de las entradas.
- Utiliza funciones para escapar caracteres especiales.

Principio de privilegios mínimos:

- Configura permisos de usuarios de base de datos para limitar accesos innecesarios.
- Uso de cortafuegos para aplicaciones web (WAF):
- Filtra solicitudes maliciosas antes de que lleguen al servidor.

8. Investiga y describe los conceptos de bases de datos seguras y cómo se pueden implementar.

Una base de datos segura es aquella que implementa políticas, herramientas y configuraciones para proteger los datos contra accesos no autorizados, pérdida o

alteración. Las bases de datos son el núcleo de cualquier sistema informático, por lo que asegurar su integridad y disponibilidad es fundamental.

Aspectos clave de bases de datos seguras:

Autenticación y autorización:

- Usa credenciales fuertes y únicas para cada usuario.
- Implementa roles y privilegios para controlar el acceso.

Cifrado de datos:

- En tránsito: Usa protocolos seguros como TLS/SSL para proteger los datos mientras se transfieren.
- En reposo: Cifra los archivos de la base de datos para evitar su lectura directa.

Auditorías y registros (logs):

- Realiza un seguimiento de todas las actividades en la base de datos.
- Monitorea accesos sospechosos o intentos fallidos de conexión.

Seguridad de la red:

- Configura firewalls para restringir accesos.
- Usa una red privada para los servidores de bases de datos.

Actualizaciones y parches:

- Mantén siempre actualizados el software de la base de datos y el sistema operativo.

Respaldo y recuperación:

- Realiza copias de seguridad periódicas.
- Implementa un plan de recuperación ante desastres.
- Buenas prácticas para implementar bases de datos seguras:

- Realiza revisiones regulares de seguridad.
- Evita usar cuentas administrativas para tareas regulares.
- Desactiva funciones y servicios innecesarios en el servidor.

CONCLUSIÓN

La seguridad de bases de datos es un pilar esencial en la protección de la información. Los ataques de SQL Injection demuestran cómo las malas prácticas en la validación de entradas pueden comprometer la seguridad de todo un sistema. Por otro lado, implementar políticas sólidas de seguridad, como el cifrado, la autenticación y las copias de seguridad, reduce significativamente los riesgos.

Proteger una base de datos no solo implica proteger su contenido, sino también garantizar su disponibilidad y confidencialidad. Adoptar un enfoque proactivo y usar herramientas modernas de seguridad es vital para mitigar las amenazas y cumplir con regulaciones de privacidad.

BIBLIOGRAFÍA

- Garfinkel, S. (2020). Database Security: What Hackers Don’t Want You to Know. O'Reilly Media.
- Ramakrishnan, R., & Gehrke, J. (2021). Database Management Systems. McGraw-Hill.
- OWASP Foundation. OWASP Top 10: Injection. Disponible en: https://owasp.org
- Oracle Corporation. (2023). Best Practices for Securing Databases. Oracle Documentation.
- MySQL Documentation. Security in MySQL. Disponible en: https://dev.mysql.com/doc/