



TECNOLÓGICO NACIONAL DE MÉXICO.
INSTITUTO TECNOLÓGICO DE TLAXIACO

INGENIERÍA EN SISTEMAS COMPUTACIONALES.

SEGURIDAD Y VIRTUALIZACIÓN

ACTIVIDAD:

REPORTE PRACTICA 3

BASES DE DATOS SEGURAS

INTEGRANTES DE EQUIPO:

LUZ ARLETH LÓPEZ BAUTISTA

SAÚL LÓPEZ BAUTISTA

DOCENTE:

ING. EDWARD OSORIO SALINAS

SEMESTRE: SEPTIMO.

GRUPO: 7 US

TLAXIACO, OAXACA A 01 DE FEBRERO DEL 2024.

Tabla de contenido

1. Crea una base de datos en MySQL con una BD que contenga los siguientes campos en tres tablas diferentes.....	4
Ilustración 1 Creación de las tablas.....	4
2. Crea un script en Python que permita insertar los datos del archivo `customers-2000000.csv`	4
Ilustración 2 Importación de las librerías.....	4
Ilustración 3 Configuración de la bd.....	5
Ilustración 4 definición de la función para la conexión.....	5
Ilustración 5 Definición de insertar datos	6
Ilustración 6 creación del cursor	6
Ilustración 7 Lectura al archivo CSV	6
Ilustración 8 Mensajería impresión.....	7
Ilustración 9 Consultas SQL customers	7
Ilustración 10 Consultas insertar datos en la tabla address.....	7
Ilustración 11 Consultas SQL para insertar datos en la tabla users	7
Ilustración 12 Creacion de iteración en las filas.....	8
Ilustración 13 Creación de una tupla	8
Ilustración 14 Creación de users_tuple.....	9
Ilustración 15 Manejos de errores	9
Ilustración 16 Valores de confirmación de cambios	9
Ilustración 17 Ruta del archivo CSV.....	10
Ilustración 18 Inserción de un nuevo registro.....	10
Ilustración 19 Mensaje en consola sobre datos insertados	10
3. Crea tres usuarios en MySQL con los siguientes permisos:	11
Usuario 1: Permisos de lectura en la tabla `customers`	11
Usuario 2: Permisos de lectura y escritura en la tabla `address`	11
Usuario 3: Permisos de lectura, escritura y eliminación en la tabla `users`	11
Ilustración 20 Código de creación de primer usuario	11
Ilustración 21 Usuario creado exitosamente	11
Ilustración 22 Código para crear segundo usuario	12
Ilustración 23 Segundo usuario creado correctamente.....	12
Ilustración 24 Creación del tercer usuario	13

Ilustración 25 Usuario creado correctamente	13
Ilustración 26 Importación de las librerías.....	14
Ilustración 27 Configuración para lo de la base de datos	14
Ilustración 28 Creación de la función para conectar.....	15
Ilustración 29 creación de la función injection SQL	17
Ilustración 30 Bloque principal.....	18
Ilustración 31 Tabla de usuarios.....	18
4. Crea un backup de la base de datos `secure_db` y restaura la base de datos en un servidor diferente.....	19
Ilustración 32 Tablas de la base de datos	19
Ilustración 33 Exportación de la base de datos.....	19
Ilustración 34 Descarga de la base de datos	20
Ilustración 35 Creación de la base de datos en otro servidor	20
Ilustración 36 Importación de la base de datos	21
Ilustración 37 Creación exitosamente.....	21
Conceptos de SQL Injection y cómo se pueden prevenir.....	22
Bases de Datos Seguras: Conceptos e Implementación	23
CONCLUSION	25
BIBLIOGRAFIA.....	26

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



```
MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0010 segundos.)

USE `secure_db`;

[ Editar en línea ] [ Editar ] [ Crear código PHP ]

Error: #1046 Base de datos no seleccionada

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0276 segundos.)

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;

[ Editar en línea ] [ Editar ] [ Crear código PHP ]

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0157 segundos.)

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, `customer_id` VARCHAR(45) NOT NULL REFERENCES customers(customer_id), PRIMARY KEY (`id`)) ENGINE = InnoDB;

[ Editar en línea ] [ Editar ] [ Crear código PHP ]

MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0178 segundos.)

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`)) ENGINE = InnoDB;

[ Editar en línea ] [ Editar ] [ Crear código PHP ]
```

Ilustración 1 Creación de las tablas

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

Paso 1. Importamos las librerías necesarias las cuales se muestran a continuación

- El import pandas as pd nos ayudará a manipular lo que serían nuestros datos.
- import mysql.connector Importa el módulo mysql.connector, que proporciona la funcionalidad para conectar y manejar bases de datos MySQL desde Python.
- from mysql.connector import Error Importa la clase Error del módulo mysql.connector.

```
import pandas as pd
import mysql.connector
from mysql.connector import Error
```

Ilustración 2 Importación de las librerías

Paso 2. Configuramos la conexión a la base de datos

- Creamos db_config que contiene los parámetros necesarios para establecer una conexión con la base de datos MySQL, los parámetros incluyen el host, user, password, database

```
# Configuración de la conexión
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '', # Asegúrate de poner el password aquí
    'database': 'secure_db'
}
```

Ilustración 3 Configuración de la bd

Paso 3. Definimos la función para conectar nuestra base de datos, donde def connect_to_database() define nuestra función llamada connect_to_database que conectará nuestra base de datos usando la configuración en db_config, donde si la conexión se realizó de forma correcta nos mostrará un mensaje de conectado a la base de datos de lo contrario si algo salió mal nos mostrará un mensaje de error al conectar a la base de datos.

```
def connect_to_database():
    try:
        connection = mysql.connector.connect(**db_config)
        if connection.is_connected():
            print("Conectado a la base de datos")
            return connection
    except Error as e:
        print(f"Error al conectar a la base de datos: {e}")
        return None
```

Ilustración 4 definición de la función para la conexión

Paso 4. Definimos una función que nos permitirá insertar datos desde el archivo CSV.

- Nuestro `def insert_data_from_csv(csv_file_path)` define una función llamada `insert_data_from_csv` que toma la ruta del archivo CSV como argumento, después `connection = connect_to_database()`, llama a la función `connect_to_database` para obtener una conexión a la base de datos, donde nuestro `if connection is None` Verifica si la conexión es `None`, lo que indica que la conexión no se pudo establecer y por ultimo `return` sale de la función si no se pudo conectar a la base de datos.

```
Tabnine: Edit | Test | Explain | Document | Ask
def insert_data_from_csv(csv_file_path):
    connection = connect_to_database()
    if connection is None:
        return
```

Ilustración 5 Definición de insertar datos

Paso 5. Creamos un curso para ejecutar muestra consulta sql

```
cursor = connection.cursor()
```

Ilustración 6 creación del cursor

Paso 6. Creamos data el cual leerá el archivo CSV usando pandas

```
data = pd.read_csv(csv_file_path)
```

Ilustración 7 Lectura al archivo CSV

Paso 7. Después imprimimos un mensaje mostrando los nombres de las columnas del archivo CSV.

```
print("Columnas en el archivo CSV:", data.columns)
```

Ilustración 8 Mensajería impresión

Paso 8. Preparamos las consultas SQL para insertar datos, donde `insert_customers_query` define una consulta SQL para insertar datos en la tabla `customers`. Los valores se insertan usando marcadores de posición (`%s`).

```
insert_customers_query = """
INSERT INTO customers (customer_id, first_name, last_name, subscription)
VALUES (%s, %s, %s, %s)
"""
```

Ilustración 9 Consultas SQL customers

Paso 9. `insert_address_query` define una consulta SQL para insertar datos en la tabla `address`.

```
insert_address_query = """
INSERT INTO address (customer_id, company, city, country, phone_1, phone_2)
VALUES (%s, %s, %s, %s, %s, %s)
"""
```

Ilustración 10 Consultas insertar datos en la tabla address

Paso 10. `insert_users_query`: define una consulta SQL para insertar datos en la tabla `users`. La contraseña se establece como `'default_password'`.

```
insert_users_query = """
INSERT INTO users (customer_id, email, password)
VALUES (%s, %s, 'default_password') # Puedes cambiar 'default_password'
"""
```

Ilustración 11 Consultas SQL para insertar datos en la tabla users

Paso 11. Creamos una iteración para las filas del dataframe, donde el index es el índice de la fila, y row es una serie que contiene los datos de la fila, con el try comenzara a realizar inserciones dentro de nuestro bloque de código.

```
for index, row in data.iterrows():
    try:
        # Insertar en la tabla 'customers'
        customer_tuple = (
            row['customer_id'],
            row['first_name'],
            row['last_name'],
            row['subscription_date'],
            row['website']
        )
        cursor.execute(insert_customers_query, customer_tuple)
```

Ilustración 12 Creacion de iteración en las filas

Paso 12. Creamos una tupla para los datos de la fila para realizar su consulta.

```
# Insertar en la tabla 'address'
address_tuple = (
    row['customer_id'],
    row['Company'],
    row['City'],
    row['Country'],
    row['Phone 1'],
    row['Phone 2']
)
cursor.execute(insert_address_query, address_tuple)
```

Ilustración 13 Creación de una tupla

Paso 13. Creamos `users_tuple` para los datos de la fila para la consulta `insert_users_query`, donde `cursor.execute(insert_users_query, users_tuple)` ejecutará la consulta SQL para insertar los datos en la tabla `users`.

```
# Insertar en la tabla 'users'
users_tuple = (
    row['customer_id'],
    row['Email']
)
cursor.execute(insert_users_query, users_tuple)
```

Ilustración 14 Creación de `users_tuple`

Paso 14. Agregamos lo que sería nuestros manejos de errores durante la inserción

```
except KeyError as e:
    print(f"Error de clave en la fila {index}: {e}")
except Error as e:
    print(f"Error al insertar los datos en la fila {index}: {e}")
```

Ilustración 15 Manejos de errores

Paso 15. De igual forma agregamos valores que nos confirmarán los cambios y cerrar conexión

```
connection.commit()
cursor.close()
connection.close()
print("Datos insertados exitosamente")
```

Ilustración 16 Valores de confirmación de cambios

Paso 16. Por ultimo Definimos la ruta del archivo CSV e iniciar el proceso

```
csv_file_path = 'C:/xampp/mysql/data/secure_db/customers-2000000.csv'
insert_data_from_csv(csv_file_path)
```

Ilustración 17 Ruta del archivo CSV

Resultados

✓ Mostrando filas 0 - 0 (total de 1, La consulta tardó 0,8282 segundos.)

```
SELECT * FROM `customers` WHERE `subscription_date` > '2024-01-01';
```

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

☐ Mostrar todo | Número de filas: 25 ▼ | Filtrar filas:

Opciones extra

	id	customer_id	first_name	last_name	subscription_date	website
<input type="checkbox"/> Editar Copiar Borrar	2042515	CUST123	John	Doe	2024-09-16	example.com

⬆ ☐ Seleccionar todo *Para los elementos que están marcados:* [Editar](#) [Copiar](#) [Borrar](#) [Exportar](#)

Ilustración 18 Inserción de un nuevo registro

```
PS C:\xampp\mysql\data\secure_db>
./secure_db/conexion.py
Conectado a la base de datos
Columnas en el archivo CSV: Index(['Index', 'customer_id', 'first_name', 'last_name', 'Company', 'City',
    'Country', 'Phone 1', 'Phone 2', 'Email', 'subscription_date',
    'website'],
    dtype='object')
Datos insertados exitosamente
```

Ilustración 19 Mensaje en consola sobre datos insertados

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`

Paso 1. Creamos nuestro primer usuario de lectura en la tabla 'customer', donde especificamos el nombre del usuario el cual solo podrá conectarse desde la máquina local, después con el identified by 'password1' para que el usuario pueda autenticarse, con el grant select asignamos permisos.

```
-- Crear Usuario 1 con permisos de lectura en la tabla `customers`  
CREATE USER 'user1'@'localhost' IDENTIFIED BY 'password1';  
GRANT SELECT ON secure_db.customers TO 'user1'@'localhost';
```

Ilustración 20 Código de creación de primer usuario

The screenshot shows a MySQL command-line interface with a light blue background. It displays two successful SQL queries and their results. The first query is 'CREATE USER IF NOT EXISTS 'user1'@'localhost' IDENTIFIED BY 'password1';' and the second is 'GRANT SELECT ON secure_db.customers TO 'user1'@'localhost';'. Both queries are preceded by a green checkmark icon and a message indicating that MySQL returned an empty set of values (zero columns) and the execution time. Below the first query, there is a warning message: 'Note: #1973 Can't create user 'user1'@'localhost'; it already exists'.


```
✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0005 segundos.)  
  
CREATE USER IF NOT EXISTS 'user1'@'localhost' IDENTIFIED BY 'password1';  
  
[ Editar en línea ] [ Editar ] [ Crear código PHP ]  
  
⚠ Note: #1973 Can't create user 'user1'@'localhost'; it already exists  
  
✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0016 segundos.)  
  
GRANT SELECT ON secure_db.customers TO 'user1'@'localhost';
```

Ilustración 21 Usuario creado exitosamente

Paso 2. Creamos nuestro segundo usuario de la misma forma que el primero pero asignándole el otro campo que solicita de lectura en la tabla 'address', donde especificamos el nombre del usuario el cual solo podrá conectarse desde la máquina local, después con el identified by 'password2' para que el usuario pueda autenticarse, con el grant select asignamos permisos.


```
CREATE USER IF NOT EXISTS 'user2'@'localhost' IDENTIFIED BY 'password2';  
GRANT SELECT, INSERT, UPDATE ON secure_db.address TO 'user2'@'localhost';
```

Ilustración 22 Código para crear segundo usuario

 MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0016 segundos.)

```
CREATE USER IF NOT EXISTS 'user2'@'localhost' IDENTIFIED BY 'password2';
```

[Editar en línea] [Editar] [Crear código PHP]

 MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0016 segundos.)

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

[Editar en línea] [Editar] [Crear código PHP]

Ilustración 23 Segundo usuario creado correctamente

Paso 2. Creamos nuestro tercer usuario de la misma forma que el primero pero asignándole el otro campo que solicita de lectura en la tabla 'users', donde especificamos el nombre del usuario el cual solo podrá conectarse desde la máquina local, después con el identified by 'password3' para que el usuario pueda autenticarse, con el grant select asignamos permisos.

```
CREATE USER IF NOT EXISTS 'user3'@'localhost' IDENTIFIED BY 'password3';  
GRANT SELECT, INSERT, UPDATE, DELETE ON secure_db.users TO 'user3'@'localhost';  
FULL PRIVILEGES;
```

Ilustración 24 Creación del tercer usuario

✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0007 segundos.)

```
CREATE USER IF NOT EXISTS 'user3'@'localhost' IDENTIFIED BY 'password3';
```

[Editar en línea] [Editar] [Crear código PHP]

⚠ Note: #1973 Can't create user 'user3'@'localhost'; it already exists

✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0031 segundos.)

```
GRANT SELECT, INSERT, UPDATE, DELETE ON secure_db.users TO 'user3'@'localhost';
```

[Editar en línea] [Editar] [Crear código PHP]

✓ MySQL ha devuelto un conjunto de valores vacío (es decir: cero columnas). (La consulta tardó 0,0046 segundos.)

```
FLUSH PRIVILEGES;
```

[Editar en línea] [Editar] [Crear código PHP]

Ilustración 25 Usuario creado correctamente

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.

Paso 1. Importamos las librerías que vamos a utilizar.

```
import mysql.connector
from mysql.connector import Error
```

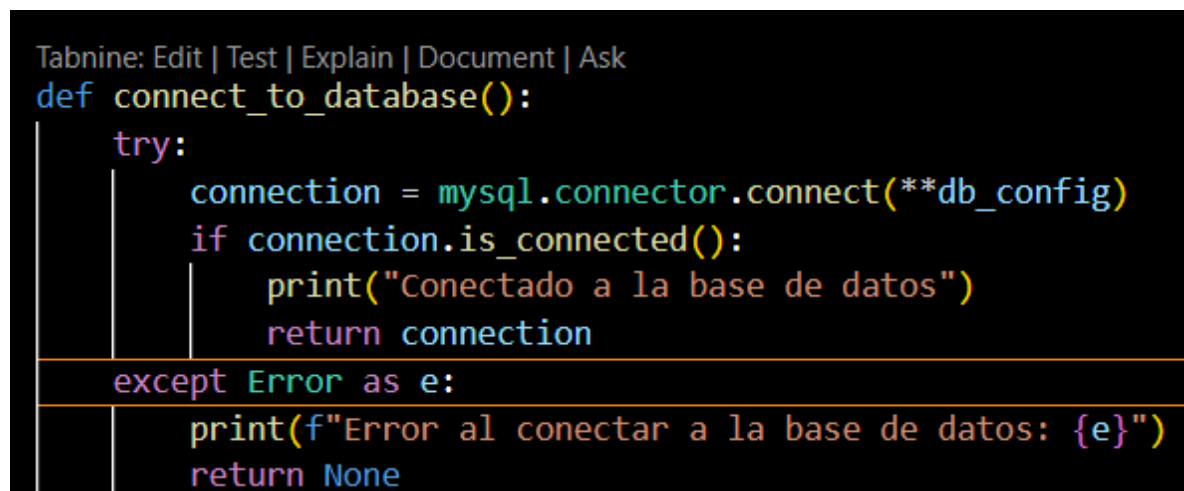
Ilustración 26 Importación de las librerías

Paso 2. Configuramos la conexión a nuestra base de datos

```
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '',
    'database': 'secure_db'
}
```

Ilustración 27 Configuración para lo de la base de datos

Paso 3. Creamos una función para conectar nuestra base de datos, donde, `mysql.connector.connect(db_config)` Intentará establecer una conexión a la base de datos usando los parámetros del diccionario `db_config`, después `connection.is_connected()`, Verifica si la conexión fue exitosa donde mandamos a mostrar un mensaje `print("Conectado a la base de datos")` si la conexión fue exitosa el cual el `return connection` va a devolver si el objeto de conexión si la conexión es exitosa y si no de lo contrario nos mostrara un mensaje de error de conexión fallida.

A screenshot of a code editor with a dark background. At the top, there is a menu bar with the text 'Tabnine: Edit | Test | Explain | Document | Ask'. Below the menu bar, the code for a Python function is displayed. The function is named 'connect_to_database' and is defined with a 'try' block. Inside the 'try' block, there is a line 'connection = mysql.connector.connect(**db_config)', followed by an 'if' statement 'if connection.is_connected():'. Inside the 'if' block, there is a 'print' statement 'print("Conectado a la base de datos")' and a 'return' statement 'return connection'. After the 'try' block, there is an 'except' block 'except Error as e:' which contains a 'print' statement 'print(f"Error al conectar a la base de datos: {e}")' and a 'return' statement 'return None'. The code is color-coded: 'def' is blue, 'try:' is pink, 'connection' is green, 'mysql.connector.connect' is green, '**db_config' is yellow, 'if' is blue, 'connection.is_connected()' is green, 'print' is green, 'Conectado a la base de datos' is orange, 'return' is pink, 'except' is blue, 'Error as e:' is pink, 'print' is green, 'f"Error al conectar a la base de datos: {e}"' is orange, and 'return None' is pink.

```
Tabnine: Edit | Test | Explain | Document | Ask
def connect_to_database():
    try:
        connection = mysql.connector.connect(**db_config)
        if connection.is_connected():
            print("Conectado a la base de datos")
            return connection
    except Error as e:
        print(f"Error al conectar a la base de datos: {e}")
        return None
```

Ilustración 28 Creación de la función para conectar

Paso 4. Ahora creamos nuestra función `injection SQL`, donde, `perform_sql_injection()`, esta función maneja la conexión a la base de datos y realiza una consulta SQL que simula una inyección SQL.

- `connection = connect_to_database()`: Llama a la función `connect_to_database` para obtener una conexión a la base de datos.
- `if connection is None`: Verifica si la conexión falló. Si es así, la función termina.
- `cursor = connection.cursor()`: Crea un cursor para ejecutar consultas SQL en la base de datos.
- `sql_injection_query = """SELECT * FROM users WHERE '1'='1';"""`: Define una consulta SQL que siempre es verdadera ('1'='1'). Esto selecciona todas las filas de la tabla `users`, demostrando una inyección SQL simple.
- `cursor.execute(sql_injection_query)`: Ejecuta la consulta SQL.
- `results = cursor.fetchall()`: Recupera todas las filas resultantes de la consulta.
- `print("Resultados de la consulta SQL inyectada:")`: Imprime un encabezado para los resultados.
- `for row in results: print(row)`: Imprime cada fila de los resultados.
- `except Error as e`: Captura y maneja errores durante la ejecución de la consulta.
- `print(f"Error al ejecutar la consulta: {e}")`: Imprime un mensaje de error si la consulta falla.
- `cursor.close()`: Cierra el cursor.
- `connection.close()`: Cierra la conexión a la base de datos.


```

Tabnine: Edit | Test | Explain | Document | Ask
def perform_sql_injection():
    connection = connect_to_database()
    if connection is None:
        return

    cursor = connection.cursor()

    sql_injection_query = """
    SELECT * FROM users WHERE '1'='1';
    """

    try:
        cursor.execute(sql_injection_query)
        results = cursor.fetchall()

        print("Resultados de la consulta SQL inyectada:")
        for row in results:
            print(row)

    except Error as e:
        print(f"Error al ejecutar la consulta: {e}")

    cursor.close()
    connection.close()

```

Ilustración 29 creación de la función injection SQL

Paso 5. Creamos un bloque principal el cual llama a la función de la inyección SQL

```
if __name__ == "__main__":  
    perform_sql_injection()
```

Ilustración 30 Bloque principal

Resultados de la inyección de la tabla user en consola

```
(1682384, 'jaclyn27@clements.com', 'default_password', 'fCbE8FAAf1D1CBC')  
(1682385, 'sanfordeduardo@baird.net', 'default_password', '046e8Bdf688CE3e')  
(1682386, 'kaneshelby@harmon.com', 'default_password', '488a5bfFEE74064')  
(1682387, 'gregg23@olsen-harmon.com', 'default_password', 'cCBa62dBe5e0966')  
(1682388, 'candice78@benjamin-cisneros.net', 'default_password', 'BAAd4D57162eD079')  
(1682389, 'lutzmegan@roberson.com', 'default_password', 'd04A2CDfc5805b8')  
(1682390, 'walter19@eaton.com', 'default_password', '0976f72B9E76f80')  
(1682391, 'alee@villanueva.com', 'default_password', '6A5DB6BB8D33DA4')  
(1682392, 'kle@sawyer-villa.net', 'default_password', '6DdB3FB0CC8B5ed')  
(1682393, 'rushariana@ramos.info', 'default_password', '12858C5BF27D1DB')  
(1682394, 'ealvarez@sanford.com', 'default_password', '798C531bD660e2A')  
(1682395, 'sethbuckley@hawkins.info', 'default_password', '2e8BcC495de41bD')  
(1682396, 'lmoran@alvarez.info', 'default_password', 'a7eFAf58c6ddad7')  
(1682397, 'collinbanks@arellano.biz', 'default_password', '9Eebde8d724bd05')
```

Ilustración 31 Tabla de usuarios

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

Paso 1. En nuestra imagen podemos observar las tablas de nuestra base de datos donde proseguiremos a realizar lo que sería nuestro backup.

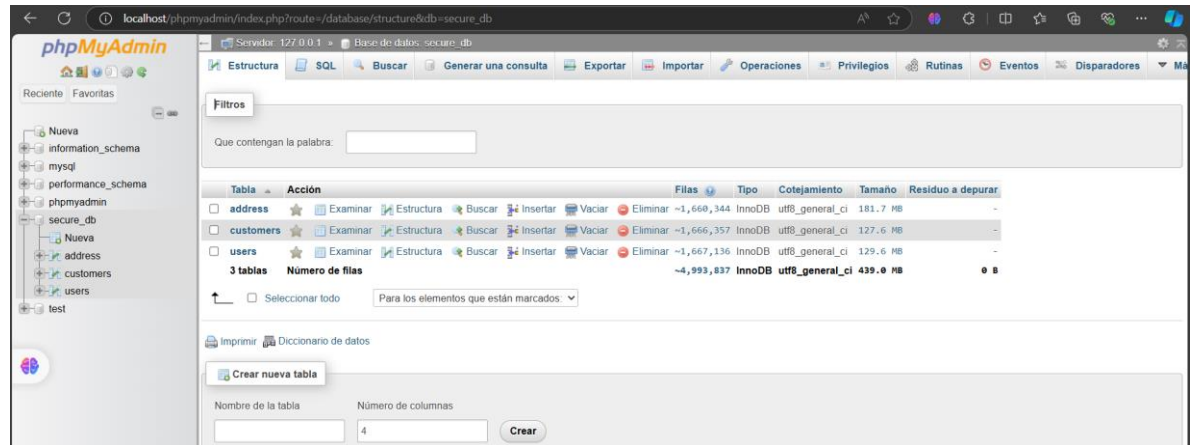


Ilustración 32 Tablas de la base de datos

Paso 2. Nos dirigimos a la opción de exportar, una vez estando ahí nos dirigimos donde dice exportar.



Ilustración 33 Exportación de la base de datos

Paso 3. Vemos que se descargó la base de datos en archivo block de notas.

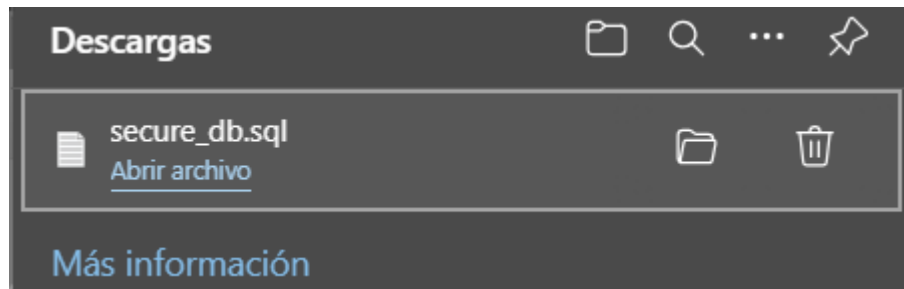


Ilustración 34 Descarga de la base de datos

Paso 4. El archivo lo pasamos a otra computadora para poder obtener su base de datos, escribimos el nombre de la base de datos y le damos click en crear.

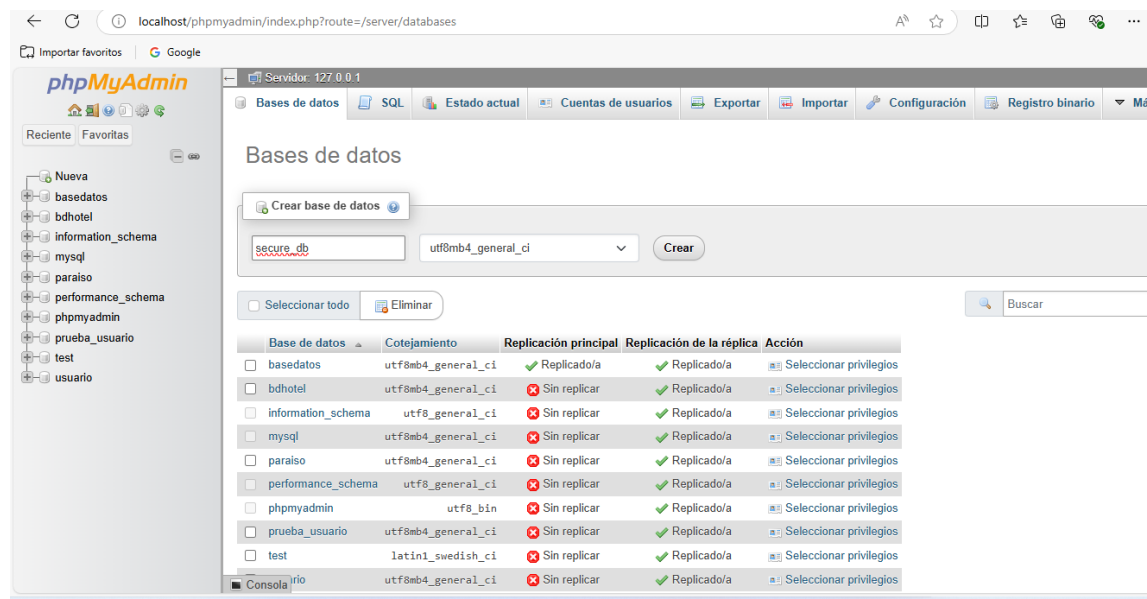


Ilustración 35 Creación de la base de datos en otro servidor

Paso 5. Nos dirigimos a la opción de importar y seleccionamos el archivo en block de notas y damos en abrir.

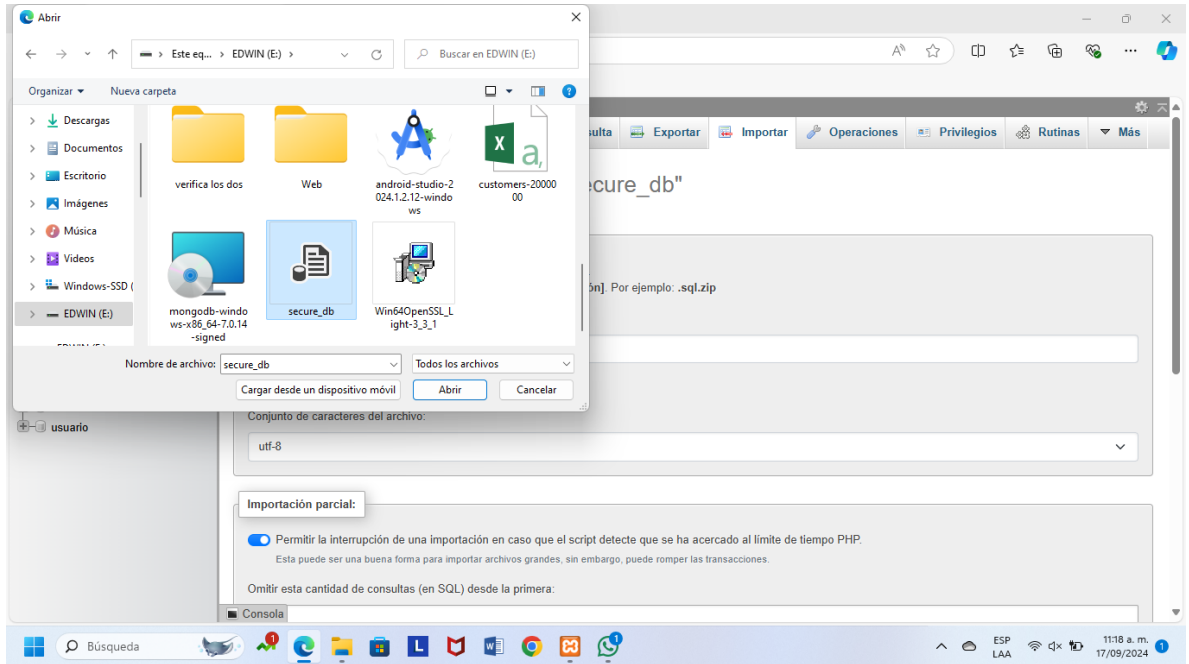


Ilustración 36 Importación de la base de datos

Paso 6. Por ultimo podemos notar que hemos creado todo.

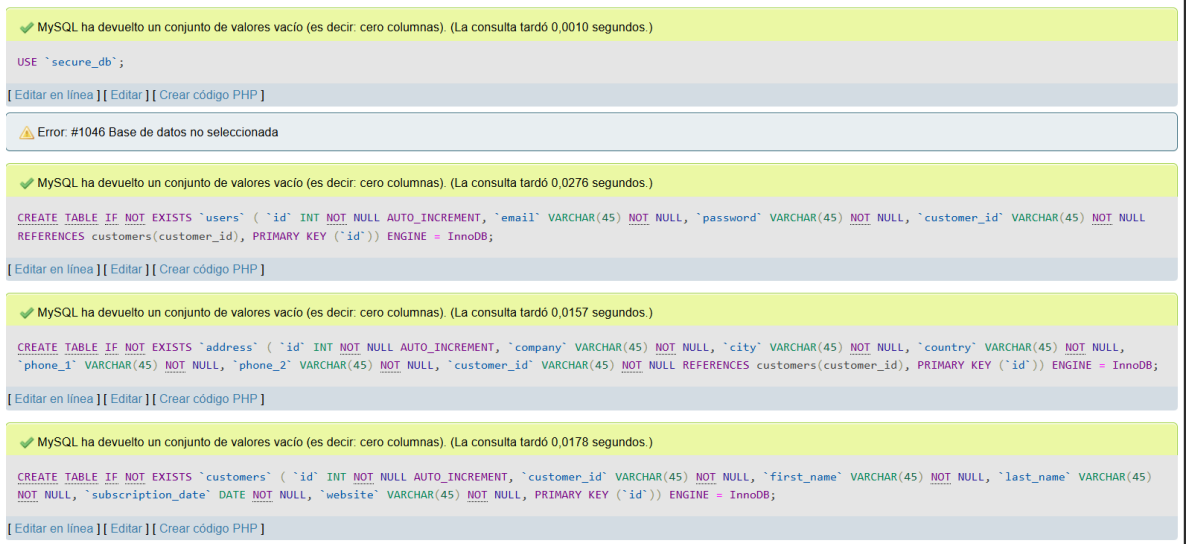


Ilustración 37 Creación exitosamente

Conceptos de SQL Injection y cómo se pueden prevenir.

SQL INJECTION: CONCEPTO Y PREVENCIÓN

El SQL Injection es una vulnerabilidad en aplicaciones que interactúan con bases de datos, permitiendo a un atacante inyectar código SQL malicioso a través de las entradas del usuario. Esto sucede cuando la aplicación no valida correctamente los datos ingresados, lo que permite al atacante modificar la consulta y acceder, manipular o eliminar datos confidenciales.

Ejemplo de SQL Injection.

Imagina una aplicación web con un formulario de inicio de sesión que genera la siguiente consulta SQL para verificar las credenciales del usuario:

```
SELECT * FROM users WHERE username = 'usuario' AND password = 'contraseña';
```

Si la entrada del usuario no es validada, un atacante podría ingresar la siguiente cadena en el campo del nombre de usuario:

```
'OR '1' = '1
```

Esto modificaría la consulta a:

```
SELECT * FROM users WHERE username = " OR '1' = '1' AND password = 'contraseña';
```

La condición `'1' = '1'` siempre es verdadera, por lo que el atacante obtendría acceso a todos los registros de la tabla `users`, evadiendo la autenticación.

MEDIDAS DE PREVENCIÓN:

1. **Consultas Parametrizadas:** Usar sentencias preparadas que separan los datos de la lógica SQL, evitando que el código malicioso se interprete como parte de la consulta.

Ejemplo en Java:

```
String query = "SELECT * FROM users WHERE username = ? AND password = ?";
```

```
PreparedStatement stmt = connection.prepareStatement(query);
```

```
stmt.setString(1, username);
```

```
stmt.setString(2, password);
```

```
ResultSet rs = stmt.executeQuery();
```

2. **Validación de Entradas:** Restringir los tipos de datos permitidos y validar todos los valores ingresados por el usuario.

3. **Escapar Caracteres Especiales:** Asegurarse de que caracteres como `", `", `;`, y `--` se traten como datos y no como parte del código SQL.

4. **Principio de Privilegios Mínimos:** Asignar los permisos mínimos necesarios a las cuentas de usuario de la base de datos, evitando que un atacante acceda o modifique datos críticos.

Bases de Datos Seguras: Conceptos e Implementación

Las bases de datos seguras protegen la confidencialidad, integridad y disponibilidad de los datos almacenados. La seguridad de las bases de datos es crucial para prevenir accesos no autorizados, corrupción o pérdida de datos.

Estrategias para Bases de Datos Seguras:

1. **Cifrado de Datos:** Protege los datos tanto en reposo como en tránsito mediante cifrado.

- 2. Control de Acceso Basado en Roles (RBAC):** Limita los permisos según los roles de los usuarios, asegurando que cada persona tenga acceso solo a lo que necesita.
- 3. Autenticación Fuerte:** Usar autenticación multifactor (MFA) y contraseñas seguras.
- 4. Políticas de Contraseñas:** Exigir contraseñas robustas y cifrarlas con algoritmos seguros.
- 5. Actualización y Parcheo:** Mantener el sistema al día para corregir vulnerabilidades.
- 6. Prevención de Inyección SQL:** Utilizar consultas parametrizadas y validar entradas.
- 7. Copias de Seguridad:** Implementar y probar regularmente sistemas de respaldo para recuperación de datos.
- 8. Monitoreo y Detección:** Vigilar actividades sospechosas en la base de datos.
- 9. Firewalls y Segmentación de Redes:** Restringir y aislar el acceso al servidor de bases de datos.
- 10. Principio de Privilegios Mínimos:** Asignar permisos mínimos necesarios a cada usuario o aplicación.

CONCLUSION

La seguridad en las bases de datos es esencial para proteger la integridad, confidencialidad y disponibilidad de la información crítica de una organización. Con el incremento de los ataques cibernéticos y las crecientes amenazas a la seguridad de los datos, implementar una estrategia robusta de seguridad de bases de datos es más importante que nunca.

Las medidas clave para asegurar una base de datos incluyen el cifrado de datos, el control de acceso basado en roles, la autenticación fuerte, la aplicación de políticas rigurosas de contraseñas, y el mantenimiento regular mediante parches y actualizaciones. Además, técnicas como el uso de consultas parametrizadas para prevenir inyecciones SQL, la realización de copias de seguridad periódicas y la monitorización continua de actividades ayudan a proteger contra accesos no autorizados y posibles violaciones de datos.

Implementar estas prácticas no solo minimiza el riesgo de ataques y pérdidas de datos, sino que también fortalece la postura de seguridad general de la organización. Al seguir estos principios, las empresas pueden garantizar que sus bases de datos permanezcan seguras y resilientes frente a las amenazas, asegurando la confianza en la integridad de su información y la continuidad de sus operaciones.

BIBLIOGRAFIA.

[¿Qué es SQL injection y cómo evitar ataques SQL injection? \(one.com\)](#)

[blog.lacnic.net](#)

[www.deltaprotect.com](#)

[ibm.com/mx-es/topics/database-security](#)

[paessler.com/es/it-explained/database](#)