Universidad de Guadalajara

Centro Universitario de los Valles

Tecnologías de la Información

Desarrollo de Bases de Datos para Web

Jesus Emmanuel Santos Chávez 221341606

TEMA: repaso bases de datos, con Mysql - Python (Crud de BD MySql - Python)

Maestro: Adolfo Castillo Chavarín

¿Para qué se utiliza mysql.conector en Python?

a) Para instalar MySQL.

b) Para conectar Python con bases de datos SQLite.

c) Para conectar Python con bases de datos MySQL.

d) Para generar reportes en PDF desde Python

La respuesta es la C ya que se utiliza para crear la conexión a una base de datos pidiendo como parámetros el nombre el usuario y la contraseña y funciona para cuando se haga la consulta se utilice y se conecte primero a la base de datos y haga la instrucción

```
print("Para continuar necesitamos el nombre de la base de datos")
    baseName = str(input("Nombre: "))
    DB = BaseDeDatos(baseName)
    print("Intentando Conectarse a la base de datos: ", end="")

    try:
        mysql = mysql.connector.connect(user='root', password='',
host='localhost', database=baseName)
        print("Conexion Exitosa a la base de datos: {}
!!!!".format(baseName))
    except Error as err:
        print("Error al conectar:", err)

    while estado == False:
        menu()
        print()
```

Se crea la conexión en la variable Mysql con el nombre del host, las credenciales y el nombre de la base de datos

```
cursor=mysql.cursor()
            cursor.execute("DROP TABLE IF EXISTS usuario1")
```

```
            cursor.execute("CREATE TABLE usuario1 (Idusuario INT
PRIMARY KEY AUTO_INCREMENT, \
                    nombres VARCHAR(25), apellidoPaterno varchar
(25),apellidoMaterno varchar (25), user varchar (10), pwd varchar
(10))")
            print("La tabla ha sido creada");
```

y aquí se utiliza el cursor de mysql para que la consulta se haga en la base de datos que conectamos.

Antes de ejecutar comandos SQL a través de mysql.connector, ¿qué objeto se debe crear?

a) Cursor

b) Commit

c) Execute

d) Connector

La A ya que el cursor se utiliza para ejecutar consultas de sql

```
def crearTabla(self, mysql):
    #with mysql:
        try:
            cursor=mysql.cursor()
            cursor.execute("DROP TABLE IF EXISTS usuario1")
            cursor.execute("CREATE TABLE usuario1 (Idusuario INT
PRIMARY KEY AUTO_INCREMENT, \
                    nombres VARCHAR(25), apellidoPaterno varchar
(25),apellidoMaterno varchar (25), user varchar (10), pwd varchar
(10))")
            print("La tabla ha sido creada");
```

Después de realizar operaciones de inserción, actualización o eliminación, ¿qué método se debe invocar para asegurarse de que los cambios se reflejen en la base de datos?

a) cursor.save()

b) cursor.run()

c) conexion.commit()

d) conexion.execute()

La C ya que commit se utiliza para hacer que los cambios se hagan de manera definitiva, esto por si en algún caso se cierra la conexión antes de tiempo commit aplica los cambios

```
def eliminarDatos(self, mysql):
        idEliminado = int(input("Dame id del usuario:"))

        #with mysql:
        try:
            cursor=mysql.cursor()
            sql='DELETE FROM usuario1 WHERE
idusuario=("%i")'%(idEliminado)
            cursor.execute(sql)
            mysql.commit()
            print ("eliminado correctamente")
        except Error as err:
            print ("error %s" %err )
```

¿Qué método del objeto cursor se utiliza para ejecutar comandos SQL?

a) run()

b) commit()

c) execute()

d) open()

La c execute por que ahí lo dice, ejecutar la instrucción

```
cursor.execute("SELECT * FROM usuario1")
```

Si se quiere recuperar muchos resultados de una consulta, en lugar de usar cursor.fetchone(), ¿qué método se debe utilizar?

a) cursor.getall()

b) cursor.queryall()

c) cursor.fetchall()

d) cursor.results()

La c ya que el fetchAll significa que rescata todos los datos en forma de arreglo

```
cursor.execute("SELECT * FROM usuario1")
resultados = cursor.fetchall()
for fila in resultados:
    print(fila)
```

¿Cómo se gestiona la configuración de la conexión a la base de datos en mysql.connector?

a) Mediante variables globales en Python.

b) Pasando argumentos directamente en el método execute().

c) Mediante un archivo de configuración externo.

d) Proporcionando argumentos como host, user, password al método connect().

La D ya que connect pide los parámetros para poderse a conectar a la base de datos y al host, sin este no se puede utilizar los demás elementos de manipulación de mysql de la libreia de sql de pyhton

```
 try:
        mysql = mysql.connector.connect(user='root', password='',
host='localhost', database=baseName)
        print("Conexion Exitosa a la base de datos: {}
!!!!".format(baseName))
    except Error as err:
        print("Error al conectar:", err)
```

Si se encuentra un error en una consulta SQL ejecutada desde Python, mysql.connector lo manejará lanzando:

a) Una alerta en consola.

b) Un objeto MySQLWarning.

c) Un objeto MySQLError.

d) Nada, simplemente no ejecutará el comando.

La c ya que será un error y este es del MYSQL, no aparece en la consola ya que retorna la variable mas no con el método print()

```
except Error as err:
print ("error %s" %err )
```

¿Cuál de las siguientes opciones no es una buena práctica cuando se trabaja con bases de datos y Python?

a) Cerrar la conexión después de terminar todas las operaciones.

b) Usar sentencias preparadas para evitar la inyección SQL.

c) Capturar excepciones al intentar conectarse a la base de datos.

d) Incorporar directamente las entradas del usuario en las consultas SQL sin validación.

La D ya que estaremos propensos a ataques sql o errores en tiempo de ejecución

```
        consulta = "SELECT * FROM usuario1 WHERE pwd = %s"
```

aqui se utiliza el %s como un control para que no ocurra ningún inyección sql

```
cursor.execute(consulta, (pwd,))
```

Al trabajar con mysql.connector, si no se cierra el cursor o la conexión explícitamente, ¿qué podría suceder?

a) Se genera automáticamente un respaldo de la base de datos.

b) Se pueden dejar conexiones abiertas innecesariamente.

c) Se convierte automáticamente la base de datos a SQLite. d) No tiene ningún efecto.

La b ya que dejar las conexiones abiertas innecesariamente ocasiona problemas de memoria y es probable que si se sigue ejecutanto en cualquier momento ya no permita las conexiones

```
try:
            cursor=mysql.cursor()
            cursor.execute("DROP TABLE IF EXISTS usuario1")
            cursor.execute("CREATE TABLE usuario1 (Idusuario INT
PRIMARY KEY AUTO_INCREMENT, \
                        nombres VARCHAR(25), apellidoPaterno varchar
(25),apellidoMaterno varchar (25), user varchar (10), pwd varchar
(10))")
            print("La tabla ha sido creada");
        except Error as err:
            print("Something went wrong: {}".format(err))
            mysql.close()
            sys.exit(1)
```

¿Qué ventaja principal ofrece mysql.connector sobre otros conectores de base de datos en Python?

a) Está construido específicamente para MySQL, por lo que ofrece una compatibilidad completa con todas las características de MySQL.

b) Puede conectarse a cualquier tipo de base de datos, no solo MySQL.

c) Acelera las consultas SQL automáticamente.

d) Traduce automáticamente las consultas SQL al idioma del servidor de base de datos.

La A ya que esta echa especifcamente para SQL, ósea que funcionara correctamente con todas las necesidades a cubrir en tus consultas, se podría decir que es muy compatible,

```
 cursor=mysql.cursor()
            sql='INSERT INTO usuario1 VALUES("%i",
"%s","%s","%s","%s","%s")'%(idusuario,nombres,ap,am,user,pwd)
            cursor.execute(sql)
            mysql.commit()
            print ("registrado correctamente")
```