CONEXIÓN A MYSQL DESDE PYTHON

En MySQL Workbench ejecutar:
ALTER USER 'root'@'localhost' IDENTIFIED BY 'AlumnaAdalab' PASSWORD EXPIRE NEVER;
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'AlumnaAdalab';

En VS Code:
PASO 1:
Importar **librería** para la conexión con MySQL:

```
import mysql.connector
from mysql.connector import errorcode
```

PASO 2:
Crear la **conexión** con los argumentos y crear casos para cada tipo de error con **errorcode**. Para solo conexión, sin conocer errores, usar código después del try.

```
try:
  cnx = mysql.connector.connect(user='root', password='AlumnaAdalab',
                                host='127.0.0.1',
                                database='tienda', ejemplo de bbdd
                                auth_plugin ='mysql_native_password')
except mysql.connector.Error as err:
  if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
    print("Something is wrong with your user name or password")
  elif err.errno == errorcode.ER_BAD_DB_ERROR:
    print("Database does not exist")
  else:
    print(err)
else:
   cnx.close() Solo cuando la queramos cerrar

print(cnx) Muestra la conexión.


mycursor = cnx.cursor()
```

PASO 3: Realizar queries

cursor(): para crear un objeto de cursor y realizar consultas en una bbdd en SQL.

execute(): enviar una consulta al servidor y ejecutarla. Cualquier query: crear tablas, alterar tablas, insertar datos...

3.1 Crear una **bbdd**

Paso 2 (sin database)

```
mycursor.execute("CREATE DATABASE BD_pruebas")
```

3.2 Crear una **tabla** dentro de una bbdd

Primero seleccionar la bbdd a usar si no se pone en la conexión(paso1):

```
mycursor.execute(USE nombre_bbdd)
```

Para crear tabla

```
mycursor = cnx.cursor()
mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")
```

3.3 Visualización de datos:

```
query = ("""SELECT city, state FROM customers
        WHERE customer_number BETWEEN 121 AND 124""") ejemplo de query
mycursor.execute(query)
```

3.4 Inserción de datos:

- Un registro:

```
query = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("Ana", "Calle 21")
mycursor.execute(query, val)
cnx.commit()
```

- Múltiples registros con una lista de tuplas**:**

```
query = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = [('Ana', 'Lowstreet 4'), ('Rocio', 'Apple st 652'), ('Juana', 'Mountain 21'), ('Pedro', 'Valley 345')]
mycursor.executemany(query, val)
cnx.commit()
```

3.5 Eliminar registros:

Si **no** hemos ejecutado **commit()** al introducir los registros:

```
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("Lucia", "Plaza 22")
mycursor.execute(sql, val)
cnx.rollback()
```

Si hemos ejecutado **commit()** al introducir los registros:

```
mycursor = cnx.cursor()
sql = "DELETE FROM customers WHERE address = 'Calle 21'"
mycursor.execute(sql)
cnx.commit()
```

3.6 Actualizar una tabla:

```
sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"
mycursor.execute(sql)
cnx.commit()
```

PASO 4: Al terminar hay que **desconectar** usando el método close():

```
cnx.close()
```

ACCESO A RESULTADOS

PASO 1: importar librerías

PASO 2: conexión a la bbbdd

```
  cnx = mysql.connector.connect(user='root', password='AlumnaAdalab',
                                host='127.0.0.1',
                                database='tienda',
                                auth_plugin ='mysql_native_password')
mycursor = cnx.cursor()
```

PASO 3:

3.1 Visualizar las **tablas** de una base de datos:

El uso de **execute()** convierte el **cursor** en un iterable al que podemos acceder

```
mycursor.execute("SHOW TABLES")
for x in mycursor:
  print(x)
```

x muestra cada tabla dentro de la bbdd tienda.

3.2 Visualizar filas de una tabla. Con el **método fetchone()** se muestra el **primer registro,** la siguiente vez que se ejecute se mostrará el siguiente registro.

```
mycursor.execute("SELECT * FROM employees")
myresult = mycursor.fetchone()
```

se puede `print(myresult)`

3.3 Visualizar todas las filas de una tabla con el **método fetchall()**, devuelve una lista de tuplas, cada tupla es una fila de la tabla.

```
mycursor.execute("SELECT * FROM employees")
myresult = mycursor.fetchall()
```

Se puede print(myresult) imprime lista de tuplas

O for x in myresult print(x) iterar para ver cada resultado

PASO 4:

```
cnx.close()
```

CREAR DATAFRAME DE PANDAS DESDE SENTENCIA SQL

PASO 1: importar librerías SQL y pandas

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

PASO 2: conexión a la bbbdd

```
  cnx = mysql.connector.connect(user='root', password='AlumnaAdalab',
                                host='127.0.0.1',
                                database='tienda',
                                auth_plugin ='mysql_native_password')

mycursor = cnx.cursor()
```

PASO 3: ejecutar una query de visualización de una tabla para imprimir un DataFrame de esta

- Opción 1: mantener columnas tabla

```
sql = "SELECT * FROM employees"
df1 = pd.read_sql_query(sql, cnx)
```

cnx es paso2

- Opción 2: renombrar columnas

```
mycursor.execute("SELECT * FROM employees")
myresult = mycursor.fetchall()
df = pd.DataFrame(myresult, colums = ['ID', 'Nombre', 'Apellido', 'Email', 'Teléfono', 'Dirección', 'Ciudad', 'País']
```

→ ejecutando df se ve el DataFrame de la tabla

PASO 4: cnx.close()

GUARDAR DATOS EN CSV

Para guardar el DataFrame de df1 en un archivo CSV

df1.to_csv('fichero.csv')

MÉTODOS DE PANDAS

.head(x) x primeras filas. Sin x, por defecto 5

.tail(x) x o 5 últimas

.sample(x) x o una aleatoria

.describe() DataFrame con los principales columnas numéricas

- include='object' devuelve un DataFrame con:
  - count: total de valores no nulos en la columna
  - unique: número de valores únicos en la columna
  - top: valor más frecuente en la columna
  - freq: frecuencia del valor más común

.duplicated() serie booleana que indica si cada fila es un duplicado de una fila previamente vista

- .sum() número de filas duplicadas

.isnull() / .isna() DataFrame de booleanos con los valores nulos

- .sum() número de valores nulos por columna

.info() resumen con columnas, cantidad valores nulos en cada una y su tipo de dato

.columns muestra el nobre de las columnas del DataFrame

Para ver las tablas de una bbdd:

use information_schema;

SELECT TABLE_NAME FROM COLUMNS WHERE TABLE_SCHEMA = 'schema'

SELECT columna/* FROM COLUMNS WHERE TABLE_SCHEMA = 'schema'