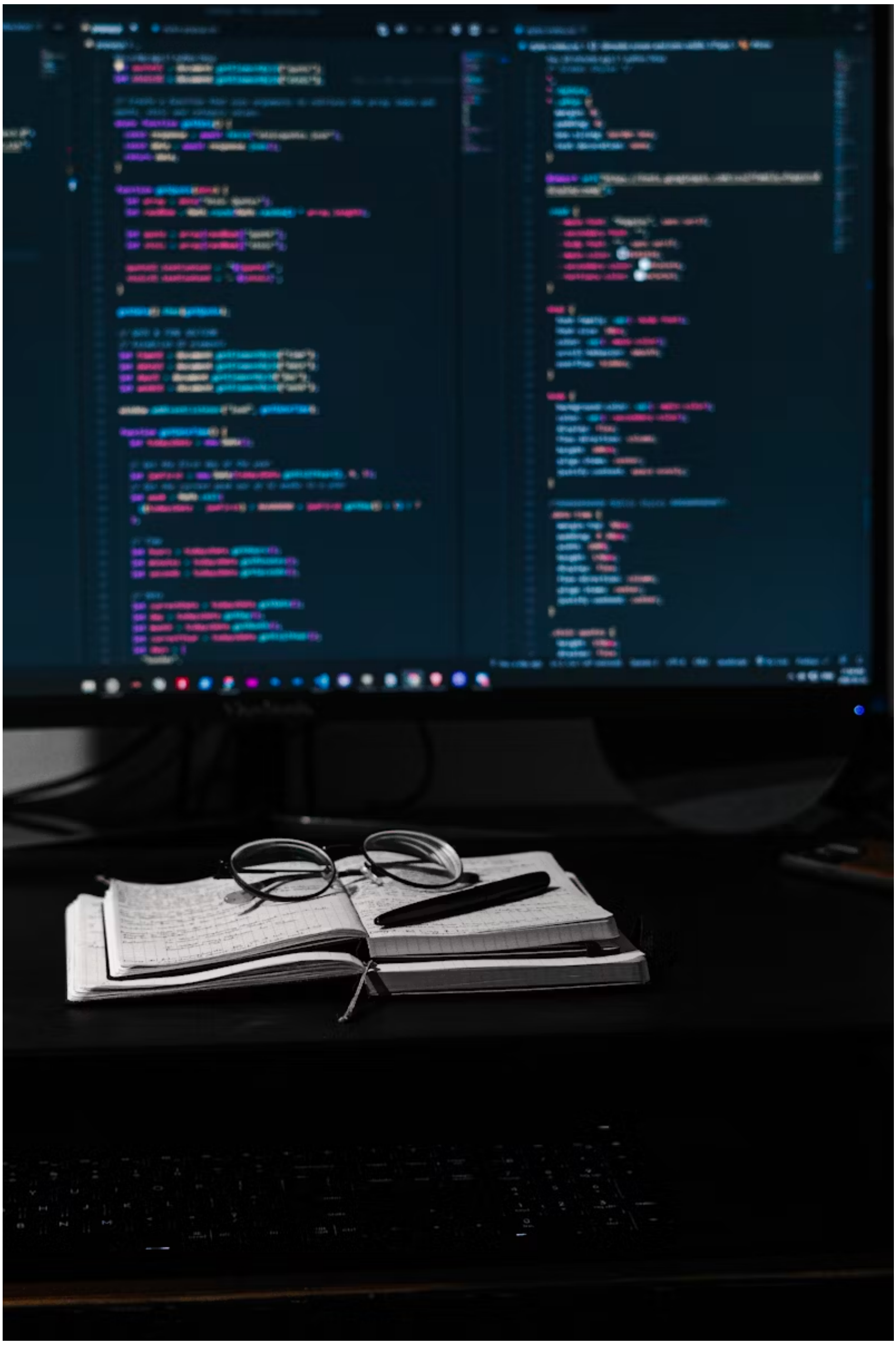
Lección 2: Driver MySQL para Python



Tiempo de ejecución: 4 horas

Planteamiento de la sesión

Materiales

- 5.1 Connecting to MySQL Using Connector/Python
- Python MySQL Insert Into Table
- How to install MySQL connector package in Python? - GeeksforGeeks

Ahora que ya somos todos unos expertos en SQL, es hora de integrarlo con Python, para ello lo primero es saber que se trabajará con el motor de MySQL y que todo lo aprendido con las consultas y manipulaciones de la base de datos es aplicable.

Para iniciar a integrar MySQL con Python se debe instalar el conector, para ello se hace uso del instalador de paquetes de python PIP. En un nuevo proyecto de VS Code que nombraremos python_mysql, vamos a crear un nuevo archivo con el nombre de mysql_connect.py.

Ahora, antes de iniciar la conexión con el servidor de MySQL, el community server, que debemos tener activo, se debe instalar el paquete, desde la terminal, utilizando el comando

python -m pip install mysql-connector-python

Al finalizar la instalación, dentro del archivo mysql_connect.py, procedemos con la importación del módulo y la configuración de la conexión con el servidor, debemos tener presente el usuario y la contraseña para poder establecer la conexión.

```
Iimport mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="your_user",
  password="your_password"
)

print(mydb)
```

Como se puede observar, el módulo mysql.connector cuenta con un método llamado connect(), al cual le podemos enviar datos adicionales como el puerto de la base de datos, por defecto es el 3306, y el nombre de la base de datos a la que nos queremos conectar, por ejemplo, sakila o w3schools.

Recordando el primer comando que ejecutamos en workbench, el comando show databases; vamos a revisar el equivalente en python

```
my_cursor = mydb.cursor()

my_cursor.execute("SHOW DATABASES")

for x in my_cursor:
  print(x)
```

Lo que hacemos aquí es crear una variable llamada my_cursor, el nombre es indiferente pero recuerde mantener el estándar visto al inicio del curso, y en la variable se asigna una instancia del método cursor() del paquete instalado, ahora podemos ejecutar sentencias SQL utilizando el comando execute.

El método execute devuelve una estructura iterable que podemos recorrer utilizando un ciclo for y finalmente imprimir en pantalla los resultados.

```
('db_xcruz',)
('heroes-app-db',)
('information_schema',)
('movies_db',)
('musimundos',)
('mwizm4g504p3z8j4',)
('mysql',)
('performance_schema',)
('sakila',)
('sampleciclo3',)
('sneakicks',)
('sys',)
```

Dado que ya contamos con un par de bases de datos de prueba, como lo son SAKILA o W3SCHOOLS, podemos modificar la cadena de conexión para que iniciemos de una vez utilizando alguna de las dos, esto es similar a ejecutar el comando USE dentro de SQL.

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="sakila"
)
```

Ahora nuestro programa al conectarse, iniciar directamente usando la base de datos de SAKILA, podemos hacer un select de la tabla actores por ejemplo y ver cómo se ejecuta la query.

```
my_cursor = mydb.cursor()

my_cursor.execute("select * from actor")

my_result = my_cursor.fetchall()

for x in my_result:
 print(x)
```

Aquí aparece un comando nuevo, el fetchall(), este comando se encarga de recuperar todas las filas o registros de la última llamada al método execute(), el resultado debería ser como el siguiente:

```
(1, 'PENELOPE', 'GUINESS', datetime.datetime(2006, 2, 15, 4, 34, 33))
(2, 'NICK', 'WAHLBERG', datetime.datetime(2006, 2, 15, 4, 34, 33))
(3, 'ED', 'CHASE', datetime.datetime(2006, 2, 15, 4, 34, 33))
(4, 'JENNIFER', 'DAVIS', datetime.datetime(2006, 2, 15, 4, 34, 33))
(5, 'JOHNNY', 'LOLLOBRIGIDA', datetime.datetime(2006, 2, 15, 4, 34, 33))
(6, 'BETTE', 'NICHOLSON', datetime.datetime(2006, 2, 15, 4, 34, 33))
(7, 'GRACE', 'MOSTEL', datetime.datetime(2006, 2, 15, 4, 34, 33))
(8, 'MATTHEW', 'JOHANSSON', datetime.datetime(2006, 2, 15, 4, 34, 33))
(9, 'JOE', 'SWANK', datetime.datetime(2006, 2, 15, 4, 34, 33))
(10, 'CHRISTIAN', 'GABLE', datetime.datetime(2006, 2, 15, 4, 34, 33))
(11, 'ZERO', 'CAGE', datetime.datetime(2006, 2, 15, 4, 34, 33))
```

el método fetchall() tiene un par de variaciones, fetchone() que trae la primera fila y fetchmany(int) que recibe un entero para retornar ese número de registros.

Otra forma de escribir la consulta es almacenarla en una variable utilizando tres comillas para la apertura y cierre (“” query “”), esto permite crear consultas en diferentes líneas y también utilizar comillas dentro del comando.

```
my_cursor = mydb.cursor();

sql = '''
SELECT *
FROM actor
WHERE upper(first_name) = "SCARLETT";
'''

my_cursor.execute(sql);

my_result = my_cursor.fetchall();

for x in my_result:
  print(x);
```

```
(81, 'SCARLETT', 'DAMON', datetime.datetime(2006, 2, 15, 4, 34, 33))
(124, 'SCARLETT', 'BENING', datetime.datetime(2006, 2, 15, 4, 34, 33))
```

Los dos últimos elementos, por ahora, que necesitamos tener presente, es la confirmación de una inserción o actualización y el cierre de la conexión al finalizar la transacción, por ejemplo dentro de la tabla actors de SAKILA quiero agregar uno nuevo, el comando sería algo como:

```
my_cursor = mydb.cursor();

sql = '''
insert into actor (first_name, last_name)
values ("Scarlett", "Johansson");
'''

my_cursor.execute(sql);
```

Necesitamos un método que permita confirmar la modificación que se aplicó, para ello, se utiliza el método commit()

```
my_cursor = mydb.cursor();

sql = '''
insert into actor (first_name, last_name)
values ("Scarlett", "Johansson");
'''
my_cursor.execute(sql);

mydb.commit();

print(my_cursor.rowcount, "record inserted.")
```

Y al final agregamos un print para identificar cuantas filas fueron afectadas o agregadas, el resultado sería algo como:

```
1 record inserted.
```

Repitiendo la consulta de encontrar registros en la tabla actor que tengan como nombre SCARLETT, debería dar como resultado un registro más:

```
(81, 'SCARLETT', 'DAMON', datetime.datetime(2006, 2, 15, 4, 34, 33))
(124, 'SCARLETT', 'BENING', datetime.datetime(2006, 2, 15, 4, 34, 33))
(201, 'Scarlett', 'Johansson', datetime.datetime(2024, 3, 15, 21, 32, 11))
```

```
my_cursor = mydb.cursor();

sql = '''
UPDATE actor
SET first_name = 'SCARLETT'
where first_name = 'Scarlett';
'''

my_cursor.execute(sql);

mydb.commit();

print(my_cursor.rowcount, "record(s) affected")
```

Ahora que ha finalizado la transacción es momento de cerrar la conexión, para ello solo es necesario hacer uso del método cose()

```
my_cursor.close();
```

Reintentando conectar

Puede que en determinadas situaciones la conexión con el servidor no ocurra a la primera y es posible configurar nuestro programa para que haga X cantidad de intentos antes de indicar que hubo un fallo. El programa ahora entonces tendrá un try…except y la conexión se realizará desde una función a la que le podemos indicar el número de intentos e incluso un retardo entre intentos.

Si simulamos un error en la contraseña o el usuario podremos ver el resultado obtenido, intentamos 3 conexiones y si no es posible conectar con el servidor, se rompe la ejecución.

```
Connection failed: %s. Retrying (%d/%d)... 1045 (28000): Access d
Connection failed: %s. Retrying (%d/%d)... 1045 (28000): Access d
Failed to connect, exiting without a connection: %s 1045 (28000):
```

El código para el archivo de conexión entonces, quedaría así:

```
import mysql.connector
import time

config = {
  "host" : "localhost",
  "user" : "root",
  "password" : "root",
  "database" : "sakila"
}

def connect_to_mysql(config, attempts=3, delay=2):
 attempt = 1
 # Implement a reconnection routine
 while attempt < attempts + 1:
    try:
        return mysql.connector.connect(**config)
    except (mysql.connector.Error, IOError) as err:
        if (attempts is attempt):
            # Attempts to reconnect failed; returning None
            print("Failed to connect, exiting without a connection: %s", err)
            return None
        print(
            "Connection failed: %s. Retrying (%d/%d)...",
            err,
            attempt,
            attempts-1,
        )
        # progressive reconnect delay
        time.sleep(delay ** attempt)
        attempt += 1
 return None

mydb = connect_to_mysql(config)
```

Utilizando parámetros

Hemos realizado varias consultas y manipulaciones sobre la base de datos, sin embargo todas han sido con los datos puestos directamente en los comandos, también es posible enviar los datos utilizando parámetros que son sustituidos al momento de ejecutar por los valores entregados. Así por ejemplo un select de los actores que se llamen SCARLETT o CUBA podría quedar de la siguiente manera:

```
my_cursor = mydb.cursor();
sql_query = '''
SELECT *
FROM actor
WHERE upper(first_name) = %s
OR upper(first_name) = %s;
'''

sql_data = ("SCARLETT", "CUBA");

my_cursor.execute(sql_query, sql_data);

my_result = my_cursor.fetchall();

for x in my_result:
  print(x);
```

Es así como los datos a insertar, modificar o consultar, pueden provenir de otra fuente y así ser dinámicos.

Para finalizar, ¿qué tal este formato?

```
Su nombre es CUBA, y se apellida OLIVIER
Su nombre es CUBA, y se apellida ALLEN
Su nombre es CUBA, y se apellida BIRCH
Su nombre es SCARLETT, y se apellida DAMON
Su nombre es SCARLETT, y se apellida BENING
Su nombre es SCARLETT, y se apellida Johansson
```

Para construirlo, solo es necesario un pequeño cambió en la impresión:

```
my_cursor = mydb.cursor();

sql_query = '''
SELECT first_name, last_name
FROM actor
WHERE upper(first_name) = %s
OR upper(first_name) = %s
ORDER BY first_name;
'''

sql_data = ("SCARLETT", "CUBA");

my_cursor.execute(sql_query, sql_data);

my_result = my_cursor.fetchall();

for (first_name, last_name) in my_result:
        print(f"Su nombre es {first_name}, y se apellida {last_name}".format(first_name, last_name));
```

Reto

Ahora qué tal un reto, se ha trabajado con unos ejercicios propuestos sobre SAKILA y de W3SCHOOL, qué tal entonces crear un programa que tome los ejercicios que se hicieron allí y los pase a funciones que se ejecutan según el usuario quiera seleccionarla utilizando un menú.

Por ejemplo, el ejercicio 1 dice que consulte todos los actores con primer nombre SCARLETT, al pulsar la opción 1 del menú, el sistema debe ejecutar la función que corresponda, recuerde cerrar la conexión al usuario finalizar el programa con la opción cero (0).

Algo adicional para el reto es que se le puede pedir al usuario un parámetro, como por ejemplo el criterio de búsqueda o el limite y dichos datos se reciben como variables que ajustan la consultas, así entonces, se tendrá una versión que buscará los actores con nombre SCARLETT pero también podría haber una en la que el usuario determina que quiere buscar, por ejemplo la opción buscar por nombre o buscar por apellido y que el usuario introduzca CUBA o WEST y que el sistema responda correctamente.

Una consulta sencilla para aplicar lo propuesto, es encontrar las películas que duren entre X y Y minutos.

Manos a la obra.