Acum că Alex a învățat cum să stabilească o conexiune cu baza de date MySQL folosind Python, este timpul să treacă la următorul pas – **executarea interogărilor SQL direct din Python**. Acest lucru înseamnă că Alex poate acum **manipula dinamic datele** fără a mai fi nevoie să introducă manual comenzile SQL în MySQL Workbench sau în terminal.

Ce vom învăța în această lecție?

În această lecție, Alex va învăța cum să folosească **Python pentru a gestiona datele într-o bază de date MySQL**, inclusiv:

- **introducerea de date noi** în baza de date folosind interogarea **INSERT**;
- **afișarea și căutarea datelor** folosind interogarea **SELECT**;
- **actualizarea datelor existente** folosind interogarea **UPDATE**;
- **ștergerea datelor** folosind interogarea **DELETE**.

Să ne amintim cunoștințele anterioare – Operațiile CRUD

Operațiunile CRUD sunt **baza lucrului cu bazele de date**. CRUD este un acronim pentru:

- **C**reate → Adăugarea de date noi în baza de date (**INSERT**).
- **R**ead → Selectarea și afișarea datelor (**SELECT**).
- **U**pdate → Modificarea datelor existente (**UPDATE**).
- **D**elete → Ștergerea datelor din baza de date (**DELETE**).

Gestionarea bazei de date prin operațiile CRUD îi permite lui Alex să creeze aplicații dinamice care interacționează cu baza de date în timp real.

Să vedem cum poate Alex să folosească Python pentru a gestiona baza de date MySQL!

Conectarea la baza de date MySQL din Python

Înainte ca Alex să poată executa interogări SQL din Python, trebuie să fie sigur că conexiunea la baza de date MySQL **a fost stabilită cu succes**.

Recapitulare

Din lecţia anterioară, ne amintim că:

- Am instalat **MySQL Connector** pentru Python.
- Am învățat cum să ne conectăm la baza de date MySQL folosind `mysql.connector.connect()` și am folosit această comandă pentru a ne conecta la baza de date **library**:

```
import mysql.connector

# Creating a connection to the MySQL database
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="library"
)

# Checking if the connection was successfully established
if(mydb == None):
  print("There is no connection to database.")
else:
  print("Connection to database is created.")
```

- Am înțeles parametrii cheie (`host`, `user`, `password`, `database`).

Dacă nu sunteți siguri cum să stabiliți conexiunea, reveniți la lecția anterioară, unde am explicat detaliat acest pas.

Acum că ne-am conectat la baza de date, este timpul să trecem la **acțiuni reale** – inserarea, căutarea, actualizarea și ștergerea datelor direct din Python!

Crearea unui cursor pentru executarea interogărilor SQL

Acum că **conexiunea la baza de date** a fost stabilită, următorul pas este **crearea unui obiect care permite executarea interogărilor SQL**. În Python, un cursor este un obiect care ne permite să trimitem comenzi SQL bazei de date și să gestionăm rezultatele interogărilor.

Cum creăm un cursor?

Foarte simplu – apelăm metoda **`cursor()`** asupra conexiunii (**`conn`**).

```
cursor = conn.cursor()
```

De ce folosim cursorul?

- Ne permite să executăm interogări SQL (INSERT, SELECT, UPDATE, DELETE).
- Ne permite să iterăm prin rezultatele interogărilor.
- Urmărește starea tranzacțiilor dacă lucrăm cu mai multe modificări în baza de date.

Cum se potriveşte în procesul de lucru cu baza de date MySQL?

1. **Stabilim conexiunea** la baza de date.
2. **Creăm un cursor** pentru a putea executa interogările.
3. **Executăm interogările SQL** folosind cursorul (ceea ce vom vedea în secțiunile următoare).
4. **Închidem cursorul și conexiunea** atunci când terminăm de lucrat cu baza de date.

Cum arată acest lucru în practică?

Să vedem **cum arată tot ce am făcut până acum** – de la stabilirea conexiunii până la crearea unui cursor care ne va permite să lucrăm cu baza de date.

```
import mysql.connector
# 1. Creating a connection to the database
conn = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="library"
)

# 2. Creating a cursor to execute queries
cursor = conn.cursor()

print("Connection established successfully, and the cursor
has been created.")
```

Ce ne permite această bibliotecă?

- Ne-am conectat la baza de date MySQL cu numele **library**.
- Am creat un **cursor** pe care îl vom folosi pentru a executa comenzi SQL.
- Am adăugat un mesaj pentru a ști că totul a mers bine.

Acum că avem un cursor, putem scrie interogări SQL direct din Python și putem gestiona datele din baza de date!

CREATE – Adăugarea unui autor nou în bază de date

Acum că avem **conexiunea cu baza stabilită și un cursor creat**, este timpul să adăugăm date noi în tabelul `Author`.

În SQL, folosim comanda **INSERT INTO** pentru a adăuga noi rânduri în tabel. În Python, acest lucru se face folosind cursorul și interogările SQL.

1. Primul mod – Interogarea INSERT directă

Cel mai simplu mod de a adăuga un autor în baza de date este să scriem direct interogarea SQL:

```
sql_query = "INSERT INTO Author (firstname, lastname) VALUES
('Harper', 'Lee')"
cursor.execute(sql_query)

# Executes the SQL query
conn.commit()

# Saves changes to the database
print("Author added successfully.")
```

Ce facem aici?

- **Scriem o interogare SQL** care adaugă un autor nou.
- **Executăm interogarea** folosind `cursor.execute()`.
- Confirmăm modificările în baza de date cu `conn.commit()`.

- **Afișăm un mesaj** care confirmă adăugarea cu succes.

Dar... acest mod nu este cel mai sigur pentru lucrul cu baza de date! Putem îmbunătăți securitatea utilizând interogări parametrizate, care previn injecțiile SQL.

2. Modul mai sigur – Interogarea INSERT cu intrarea utilizatorului

În loc să introducem datele direct în stringul SQL, le putem transmite ca parametri, prevenind astfel **injecţiile SQL**.

Cum funcționează acest lucru?

- Python solicită utilizatorului să introducă datele.
- Interogarea SQL folosește **placeholdere** (`%s`, `%s`) în loc de valori specifice.
- Valorile sunt transmise ca **tupluri**, iar Python se asigură automat că datele sunt corect formate.

Să vedem versiunea îmbunătățită a codului:

```
import mysql.connector

# Connecting to the MySQL database
conn = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="library"
)

# Creating a cursor to execute queries
cursor = conn.cursor()

# User input for author details
```

```
firstname = input("Enter the author's first name: ").strip()
lastname = input("Enter the author's last name: ").strip()

# SQL query with parameters (safe way to execute queries)
sql_query = "INSERT INTO Author (firstname, lastname) VALUES
(%s, %s)"
values = (firstname, lastname)

# Executing the SQL query
cursor.execute(sql_query, values)
conn.commit() # Saves changes to the database

print(f"Author {firstname} {lastname} successfully added to
the database.")

# Closing the database connection
cursor.close()
conn.close()
print("Database connection closed.")
```

De ce este acest mod mai bun?

- **Mai sigur** – previne injecțiile SQL.
- **Mai flexibil** – permite introducerea oricărui autor, nu doar al unui autor predefinit.
- **Mai interactiv** – utilizatorul introduce datele direct.

Mini activitate: Adăugarea unui utilizator nou în baza de date

Acum Alex vrea să adauge noi utilizatori în baza de date **library**, împreună cu informațiile despre orașul în care locuiesc.

Sarcină

1. Nivelul de bază:

- Să scriem un program Python care cere introducerea **numelui, prenumelui și ID-ului orașului**.
- Programul va adăuga utilizatorul în tabelul **User** folosind interogarea INSERT.
- La final, programul **va afișa un mesaj** de succes și va închide **conexiunea** la baza de date.

2. Nivelul avansat (provocare pentru reflecție suplimentară!):

- În loc de ID-ul orașului, utilizatorul va introduce **numele orașului**.
- Python va folosi mai întâi **interogarea SELECT** pentru a găsi ID-ul orașului în tabelul **City**.
- După obținerea ID-ului, interogarea INSERT va adăuga utilizatorul în tabelul **User**

Sfaturi bonus:

- Dacă utilizatorul introduce un oraș care nu există, anunțați-l despre eroare.
- Permiteți introducerea mai multor utilizatori într-o singură rulare a programului.

READ - Selectarea cărţilor din tabelul Book

Acum că Alex a învățat cum să adauge înregistrări noi la baza de date folosind **INSERT**, următorul pas logic este **selectarea datelor**. Interogările SQL SELECT ne permit să citim date din baza de date, iar Python ne ajută să procesăm acele date și să le afișăm utilizatorilor.

Alex vrea să găsească toate cărțile care aparțin genului **Fiction**. Să vedem cum folosește Python pentru a executa interogări SELECT și pentru a selecta datele:

```
# Defining the SQL query
sql_query = """SELECT title, published
  FROM Book
  WHERE genre_id = (SELECT genre_id FROM Genre WHERE name =
'Fiction')"""

# Executing the query
cursor.execute(sql_query)

# Fetching all results
books = cursor.fetchall()

# Displaying results
for book in books:
print(book)
```

În acest exemplu:

- `cursor.execute(sql_query)` – execută interogarea SQL;
- `fetchall()` – preia toate rezultatele din baza de date;
- `for book in books` – trece prin rezultate şi le afişează.

Cum se preiau rezultatele interogării?

Când folosim Python pentru executarea interogării **SELECT**, putem prelua rezultatele folosind mai multe metode:

Metodă	Returnează	Când se foloseşte?
`fetchall()`	Lista tuturor rândurilor (lista de tupluri)	Când aşteptăm mai multe rânduri în rezultatul interogării
`fetchone()`	Primul rând ca tuplu	Când aşteptăm mai multe rânduri în rezultatul interogării

`fetchone()[0]`	Prima valoare din primul rând	Când SQL returnează doar o singură valoare (*COUNT, SUM...*).

Tabelul 25.1. Metode în Pyhon pentru preluarea rezultatelor din interogarea SQL SELECT

Exemplu: Numărul tuturor cărţilor din bibliotecă

Dacă vrem să aflăm câte cărţi se găsesc în tabelul `Book`, folosim funcţia `COUNT()`:

```
cursor.execute("SELECT COUNT(*) FROM Book")
total_books = cursor.fetchone()[0]

# Retrieving the first value from the first row
print(f"Total number of books in the library:
{total_books}")
```

Tutorial video: 25.1. Executarea interogării din programul Python

Mini activitate: Afişarea tuturor utilizatorilor dintr-un anumit oraş

Alex vrea să găsească toţi utilizatorii bibliotecii care locuiesc în oraşul **Thompsonfurt**.

Sarcină: Să scriem un program Python care va selecta şi afişa toţi utilizatorii din acest oraş din baza de date **library**.

Sfat: Folosim interogarea `SELECT` cu clauza `WHERE` şi cu metoda `fetchall()` pentru preluarea rezultatelor.

UPDATE – Actualizarea datelor în baza de date MySQL folosind Python

Acum că Alex a învățat cum să adauge și să caute date în baza de date, următorul pas este **actualizarea înregistrărilor existente**. În lumea reală, datele se modifică frecvent – cărțile pot primi ediții noi, clienții își pot schimba adresele, iar bibliotecile își pot actualiza catalogul.

Pentru a reîncărca datele din baza de date, Alex folosește interogarea **UPDATE**.

Exemplu: Actualizarea anului de publicare a cărţii

Alex a observat că a introdus greșit anul de publicare al cărții „*To Kill a Mockingbird*”. În loc de anul de publicare original, acesta trebuie setat la **2025**. Să vedem cum va actualiza datele folosind Python:

```
import mysql.connector

# Connecting to the MySQL database
conn = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="library"
)

# Creating a cursor to execute queries
cursor = conn.cursor()
```

```
# SQL query to update the publication year
sql_query = """UPDATE Book
  SET published = 2025
  WHERE title = 'To Kill a Mockingbird'"""

# Executing the query
cursor.execute(sql_query)
conn.commit()

# Saving changes to the database
print("Data successfully updated.")

# Closing the connection
cursor.close()
conn.close()
print("Database connection closed.")
```

Explicația codului:

- **Stabilim conexiunea** cu baza de date.
- **Creăm cursorul** pentru executarea interogării SQL.
- **Definim şi executăm interogarea UPDATE** care schimbă anul de publicare al cărții „To Kill a Mockingbird”.
- Cu **metoda** `commit()` confirmăm modificările în baza de date.
- **Închidem conexiunea** când terminăm lucrul.

Mini activitate: Actualizarea informaţiilor despre utilizator

Alex doreşte să actualizeze adresa utilizatorului cu `user_id = 5` în tabelul `User`.

Sarcină: Să scriem un program Python care schimbă adresa acestui utilizator în `"123 New Street"`.

Provocare suplimentară: Să-i permitem utilizatorului să introducă **user_id** şi **o adresă nouă**, iar apoi să executăm

actualizarea în baza de date.

Sfat: Utilizarea interogărilor parametrizate pentru securitate

Inserarea valorilor direct în interogările SQL poate fi riscantă din cauza posibilelor **injecții SQL**. În schimb, se recomandă utilizarea **interogărilor parametrizate**, care sporesc securitatea și previn atacurile.

Iată cum am putea scrie o interogare **UPDATE** mai securizată:

```
sql_query = "UPDATE Book SET published = %s WHERE title = %s"
values = (2025, "To Kill a Mockingbird")
cursor.execute(sql_query, values)
conn.commit()
print("Data has been successfully updated.")
```

De ce să folosim interogările parametrizate?

- Previn injecțiile SQL.
- Îmbunătățesc performanța bazei de date atunci când aceeași interogare este executată de mai multe ori cu date diferite.
- Permit manipularea mai ușoară a datelor dinamice.

DELETE – Ştergerea datelor din baza de date MySQL folosind Python

Acum că Alex a învățat cum **să adauge, să caute și să actualizeze date**, următorul pas logic este **ștergerea înregistrărilor din baza de date**. În scenariile din lumea reală, ștergerea datelor este necesară atunci când datele devin depășite sau nu mai sunt relevante – de exemplu, atunci când o carte este eliminată din bibliotecă sau utilizatorul nu mai este activ.

Alex vrea să curețe baza de date a cărților publicate**înainte de anul 1950**.

Exemplu: Actualizarea anului de publicare a cărţii

Pentru a scăpa de datele vechi, Alex foloseşte interogarea **DELETE**:

```
import mysql.connector

# Connecting to the MySQL database
conn = mysql.connector.connect(
  host="localhost",
  user="root",
  password="root",
  database="library"
)

# Creating a cursor
cursor = conn.cursor()

# SQL query to delete books published before 1950
sql_query = "DELETE FROM Book WHERE published < 1950"

# Executing the query
cursor.execute(sql_query)
conn.commit()

# Saving changes to the database
print("Old books have been deleted from the database.")

# Closing the connection
cursor.close()
conn.close()
print("Database connection closed.")
```

Cum funcţionează acest lucru?

- **Ne conectăm la baza de date** folosind `mysql.connector.connect()`.
- **Creăm un cursor** care ne permite să executăm interogări SQL.
- **Definim o interogare DELETE**, care elimină toate cărțile publicate înainte de anul 1950.
- **Executăm interogarea și confirmăm modificările** folosind `conn.commit()`.
- Închidem conexiunea după efectuarea operației.

Notă: Când folosim **DELETE**, datele sunt șterse definitiv din baza de date și nu mai pot fi restaurate (cu excepția cazului în care există un backup).

Mini activitate: Ştergerea cărţilor pe baza intrării de utilizator

Alex vrea să creeze un program mai flexibil care permite utilizatorilor **să introducă anul**, iar apoi din baza de date să se şteargă toate cărţile publicate în anul respectiv.

Sarcină:

- Să scriem un program Python care cere de la utilizator să introducă anul.
- Apoi, programul şterge toate cărţile publicate în anul respectiv din baza de date.
- Să folosim **interogări parametrizate** pentru asigurarea securităţii aplicaţiei.

Concluzia lecţiei

În această lecție, Alex a învățat cum să folosească Python pentru a efectua operații CRUD pe o bază de date MySQL. Acum el poate:

- să introducă date noi în baza de date folosind

interogări **INSERT**;
- să afișeze și să analizeze datele utilizând interogări **SELECT**;
- să actualizeze înregistrările existente utilizând **UPDATE**;
- să șteargă datele din baza de date folosind **DELETE**.

Aceste abilități îi permit lui Alex să creeze aplicații dinamice care funcționează cu date în timp real, să automatizeze procesele și să facă lucrul cu bazele de date mai eficientă!

Următorul pas: Analiza datelor dintr-o bază de date MySQL folosind Pandas

Acum că Alex știe cum să insereze, să modifice și să șteargă date într-o bază de date MySQL folosind Python, este timpul să învețe cum să le **analizeze** și să le **vizualizeze** folosind instrumente puternice precum **Pandas** și **Matplotlib**.

Să facem următorul pas către **integrarea completă a Python-ului și a analizei datelor!**