UEH UNIVERSITY College of Technology and Design

Python Programming

Database connectivity

By: Võ Văn Hải

Email: vovanhai@ueh.edu.vn

1

Objectives

- Introduction
- Database API
- Working with SQLite
- Working with MySQL/MariaDB

2

Python database connectivity

Introduction

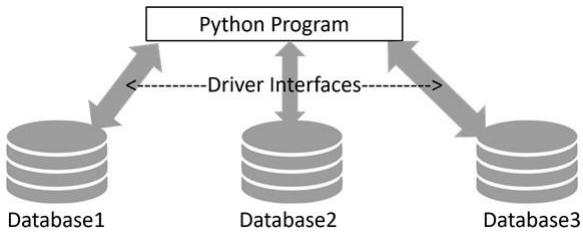
- Database access in Python is used to interact with databases, allowing applications to store, retrieve, update, and manage data consistently.
- Various relational database management systems (RDBMS) are supported for these tasks, each requiring specific Python packages for connectivity.

Relational DB	NoSQL	NewSQL
MySQL PostgreSQL Microsoft SQL Server Informix Oracle Sybase SQLite and many more...	MongoDB Apache Cassandra Neo4j CouchDB Redis Elasticsearch …	TiDB CockroachDB YugabyteDB RisingWave MatrixOne …

Python database connectivity

Database API (DB-API)

- To address this issue of compatibility, Python Enhancement Proposal (PEP) 249 introduced a standardized interface known as DB-API.
  - This interface provides a consistent framework for database drivers, ensuring uniform behavior across different database systems.
  - It simplifies the process of transitioning between various databases by establishing a common set of rules and methods.



Using SQLite with Python

Introduction

- Using SQLite with Python is easy due to the built-in sqlite3 module. The process involves
  - Connection Establishment − Create a connection object using sqlite3.connect(), providing necessary connection credentials such as server name, port, username, and password.
  - Transaction Management − The connection object manages database operations, including opening, closing, and transaction control (committing or rolling back transactions).
  - Cursor Object − Obtain a cursor object from the connection to execute SQL queries. The cursor serves as the gateway for CRUD (Create, Read, Update, Delete) operations on the database.

5

Using SQLite with Python

Memory database

```
import sqlite3
if __name__ == '__main__':
    # create connection
    con = sqlite3.connect(":memory:")  # in-memory database
    # create cursor
    cur = con.cursor()
    cur.execute("CREATE TABLE test(id INTEGER PRIMARY KEY, msg TEXT);")
    for i in range(10):
        cur.execute("INSERT INTO test VALUES (?, ?);", (i, f"Hello {i}"))
    con.commit()
    cur.execute("SELECT * FROM test")
    for row in cur: print(row)
    # save database to disk
    backup = sqlite3.connect("../../resources/sample.db")  # on disk database
    with backup:
        con.backup(backup,pages=1)
    con.close()
```

Read more at https://docs.python.org/3/library/sqlite3.html

6

Using SQLite with Python

```
import sqlite3
con = sqlite3.connect("tutorial.db")
cur = con.cursor()
cur.execute("DROP TABLE IF EXISTS movie")
cur.execute("CREATE TABLE movie(title, year, score)")

cur.execute("""
            INSERT INTO movie
            VALUES ('Monty Python and the Holy Grail', 1975, 8.2),
                   ('And Now for Something Completely Different', 1971, 7.5)
            """)
con.commit()
res = cur.execute("SELECT * FROM movie")
for row in res.fetchall(): print(row)
```

```
('Monty Python and the Holy Grail', 1975, 8.2)
('And Now for Something Completely Different', 1971, 7.5)
```

```
data = [
    ("Monty Python Live at the Hollywood Bowl", 1982, 7.9),
    ("Monty Python's The Meaning of Life", 1983, 7.5),
    ("Monty Python's Life of Brian", 1979, 8.0),
]
cur.executemany("INSERT INTO movie VALUES(?, ?, ?)", data)
con.commit()  # Remember to commit the transaction after executing INSERT.
```

```
for row in cur.execute("SELECT year, title FROM movie ORDER BY year"):
    print(row)
```

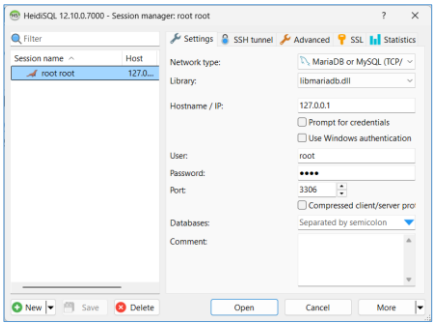
```
(1971, 'And Now for Something Completely Different')
(1975, 'Monty Python and the Holy Grail')
(1979, "Monty Python's Life of Brian")
(1982, 'Monty Python Live at the Hollywood Bowl')
(1983, "Monty Python's The Meaning of Life")
```

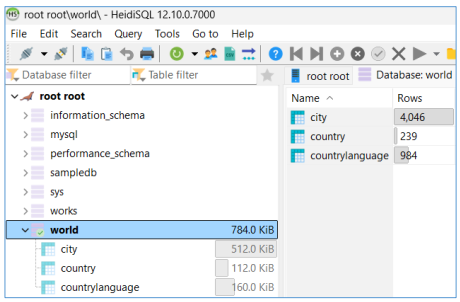

7

Connect to MySQL (MariaDB) server

Preparation

- Download & install MariaDB
  - https://mariadb.com/downloads/
- Using HeidiSQL as a database management tool





- Install the Python library for MariaDB
  - pip install mariadb

Read more at https://mariadb.com/docs/server/connect/programming-languages/python/

8

Connect to MySQL (MariaDB) server

Sample code

```
# Connect to MariaDB Platform
try:
    conn = mariadb.connect(
        user="root",
        password="root",
        host="localhost",
        port=3306,
        database="world"
    )
except mariadb.Error as e:
    print(f"Error connecting to MariaDB Platform: {e}")
    sys.exit(1)
```

```
try:
    cur = conn.cursor()
    cur.execute("INSERT INTO employees (first_name,last_name) VALUES (?, ?)", ("Maria", "DB"))
except mariadb.Error as e:
    print(f"Error: {e}")
```

Development reference:
https://mariadb.com/docs/server/connect/programming-languages/python/development/

```
# Get Cursor
cur = conn.cursor()
code = 'VNM'
cur.execute("SELECT Name, District, Population FROM city WHERE CountryCode=?", (code,))
# Print Result-set
for (name, district, population) in cur:
    print(f"Name: {name}, District: {district}, Population: {population}")
```


9

Database Manipulation

Introduction

- MariaDB Connector/Python accesses the database through a cursor, which is obtained by calling the cursor() method on the connection.
  - This cursor object provides you with an interface for performing basic operations in this section.
- The cursor provides two methods for executing SQL statements:

Method	Description
`execute()`	Executes a single SQL statement.
`executemany()`	Executes the given SQL statement for each tuple in a list.

- DML - Data Definition Language
  - ALTER TABLE, CREATE TABLE, DROP TABLE, CREATE DATABASE, and TRUNCATE TABLE.
- DDL - Data Manipulation Language
  - DELETE, INSERT, REPLACE, SELECT, and UPDATE.

10

Data Definition

```
# Instantiate Connection
try:
    conn = mariadb.connect(
        host="localhost", port=3306, user="root", password="root",
database="test",
        autocommit=True)
except mariadb.Error as e:
    print(f"Error connecting to the database: {e}")
    sys.exit(1)
```

```
def create_contacts(cur):
    try:
        sql = """CREATE TABLE IF NOT EXISTS test.contacts (
                id INT AUTO_INCREMENT PRIMARY KEY,
                first_name VARCHAR(255) NOT NULL,
                last_name VARCHAR(255) NOT NULL,
                email VARCHAR(255) NOT NULL)"""
        cur.execute(sql)
    except mariadb.Error as e:
        print(f"Error creating contacts: {e}")
        sys.exit(1)
```

#	Name	Datatype	Length/Set	Allow NULL	Default
1	id	INT	11	☐	AUTO_INCREMENT
2	first_name	VARCHAR	60	☐	No default
3	last_name	VARCHAR	50	☐	No default
4	email	VARCHAR	90	☐	No default

11

Database Manipulation (1/2)

```
# Adds a single contact
def add_contact(cur, first_name, last_name, email):
    cur.execute("INSERT INTO test.contacts(first_name, last_name, email) VALUES (?, ?, ?)",
                (first_name, last_name, email))

# Adds Multiple contacts
def add_multiple_contacts(cur, data):
    cur.executemany("INSERT INTO test.contacts(first_name, last_name, email) VALUES (?, ?,
?)", data)
```

```
# Instantiate Connection
try:
    conn = mariadb.connect(
        host="localhost", port=3306,
        user="root", password="root",
        database="test",
        autocommit=True)
except mariadb.Error as e:
    print(f"Error connecting to
the database: {e}")
    sys.exit(1)
```

#	id	first_name	last_name	email
1	1	Teo	Nguyen	teo.nguyen@example.com
2	2	Dani	Smith	dani.smith@example.com
3	3	Lee	Wang	lee.wang@example.com
4	4	Kai	Devi	kai.devi@example.com

```
# Instantiate Cursor
cur = conn.cursor()

# Call function to add a single contact
add_contact(cur,"Teo", "Nguyen","teo.nguyen@example.com")

# Initialize Data to add multiple contacts
new_contacts = [
    ("Dani", "Smith", "dani.smith@example.com"),
    ("Lee", "Wang", "lee.wang@example.com"),
    ("Kai", "Devi", "kai.devi@example.com")
]
# Call function to add multiple contacts
add_multiple_contacts(cur, new_contacts)

# Close Connection
conn.close()
```


12

Database Manipulation (2/2)

```
# Print List of Contacts
def print_contacts(cur):
    contacts = []
    # Retrieve Contacts
    cur.execute("SELECT first_name, last_name, email FROM test.contacts")
    # Prepare Contacts
    for (first_name, last_name, email) in cur:
        contacts.append(f"{first_name} {last_name} <{email}>")
    # List Contacts
    print("\n".join(contacts))
```

```
Teo Nguyen <teo.nguyen@example.com>
Dani Smith <dani.smith@example.com>
Lee Wang <lee.wang@example.com>
Kai Devi <kai.devi@example.com>
```


13

Thanks for your listening!

16