Database Systems Lab
Python DB-API

Christian Rauch

(changed: August 18, 2025)

Python DB-API

The DB-API 2.0 provides a common framework for accessing SQL databases: Classes for connecting to databases, executing queries, fetching data, and handling transactions.

https://peps.python.org/pep-0249/

Different vendor provide concrete driver libraries:

SQLite	→	sqlite3 (builit-in, no installation required)
PostgreSQL	→	psycopg2
MySQL	→	PyMySQL or mysql-connector-python
Oracle	→	cx_Oracle

Install the library (e.g., mysql-connector-python) using PIP.

```
$ python -m pip install mysql-connector-python
```

Connecting

Import the library.

```
import mysql.connector
import logging
```

Open a connection using credentials and other parameters.

```
try:
    connection = mysql.connector.connect(
        user=’user’, password=’password’,
        host=’localhost’, database=’database’
)
```

Use the connection to read or write data.

React to errors, usually by logging them.

```
except Error as e: logging.error(f"Error: {e}")
```

Close the connection after usage, typically in a finally section.

```
finally: connection.close()
```

Reading Data

Use the cursor to execute a SQL statement and process the result.

```
cursor = connection.cursor()

cursor.execute('SELECT * FROM rooms')

rows = cursor.fetchall() # or fetchmany(COUNT)
```

Convert SQL data types (e.g., CHAR, VARCHAR, FLOAT, DATE) to Python's native types (e.g., int, str, list) or user-defined types.

```
for tuple in rows:
    room = str(tuple[0])
    if tuple[1] is not None:
        capacity = int(tuple[1])
    ...
```

Mapping between SQL types and Python types requires custom code. ORM (Object-Relational Mapping) libraries like SQLAlchemy are an alternative.

Writing Data

Use the cursor to execute a SQL statement.

```python
import mysql.connector
from mysql.connector import Error

try:
    connection = mysql.connector.connect(...)
    cursor = connection.cursor()

    cursor.execute("INSERT INTO students (name, age, major) VALUES ('John Doe', 25, 'Computer Science')")

    connection.commit()

except Error as e:    print(f"Error: {e}")

finally:
    if cursor:        cursor.close()
    if connection:    connection.close()
```

Writing Binary Data

Use the cursor to execute a SQL statement to store binary data into a BLOB column.

```
...

    file_path = ...

    # read the file in binary mode
    with open(file_path, 'rb') as file:
        binary_data = file.read()

    cursor.execute("""
        INSERT INTO files (filename, file_data)
        VALUES (%s, %s)""", (..., binary_data))

...
```

Predefined Parameterized Queries

SQL statements are usually predefined with parameter placeholders to prevent SQL injection attacks.

Placeholder in queries are substituted with supplied arguments during execution:
    %s in MySQL and PostgreSQL,
    ? in SQLite

```
query = "INSERT INTO students (name, age, major) VALUES (%s, %s, %s)"

data = (’John Doe’, 25, ’Computer Science’)

cursor.execute(query, data)
```

Transactions

Default auto-commits after every statement, change with

```
connection.autocommit = False
connection.commit()
connection.rollback()
```

Transaction isolation levels (details differ in each library, check documentation). SQLite only supports SERIALIZABLE.

```
connection.set_session(...)
READ_COMMITTED
READ_UNCOMMITTED
READ_REPEATABLE_READ
SERIALIZABLE
connection.readonly = True
```

Conclusion

Thank you for your attention!