# Module 12

## B1.What database does Python use?What database API includes?

Answer:

From a construction firm to a stock exchange, every organisation depends on large databases. These are essentially collections of tables, and’ connected with each other through columns. These database systems support SQL, the Structured Query Language, which is used to create, access and manipulate the data. SQL is used to access data, and also to create and exploit the relationships between the stored data. Additionally, these databases support database normalisation rules for avoiding redundancy of data. The Python programming language has powerful features for database programming. Python supports various databases like MySQL, Oracle, Sybase, PostgreSQL, etc. Python also supports Data Definition Language (DDL), Data Manipulation Language (DML) and Data Query Statements. For database programming, the Python DB API is a widely used module that provides a database application programming interface.

Benefits of Python for database programming

There are many good reasons to use Python for programming database applications:

Programming in Python is arguably more efficient and faster compared to other languages.

Python is famous for its portability.

It is platform independent.

Python supports SQL cursors.

In many programming languages, the application developer needs to take care of the open and closed connections of the database, to avoid further exceptions and errors. In Python, these connections are taken care of.

Python supports relational database systems.

Python database APIs are compatible with various databases, so it is very easy to migrate and port database application interfaces.

DB-API (SQL-API) for Python

Python DB-API is independent of any database engine, which enables you to write Python scripts to access any database engine. The Python DB API implementation for MySQL is MySQLdb. For PostgreSQL, it supports psycopg, PyGresQL and pyPgSQL modules. DB-API implementations for Oracle are dc\_oracle2 and cx\_oracle. Pydb2 is the DB-API implementation for DB2. Python’s DB-API consists of connection objects, cursor objects, standard exceptions and some other module contents, all of which we will discuss.

## B2.Explain benefit of python database programming

Answer:

There are many good reasons to use Python for programming database applications:

Programming in Python is arguably more efficient and faster compared to other languages.

Python is famous for its portability.

It is platform independent.

Python supports SQL cursors.

In many programming languages, the application developer needs to take care of the open and closed connections of the database, to avoid further exceptions and errors. In Python, these connections are taken care of.

Python supports relational database systems.

Python database APIs are compatible with various databases, so it is very easy to migrate and port database application interfaces.

## B2.What is PyMySQL in Python?

Answer:

PyMySQL is a pure-Python MySQL client library, based on PEP 249. Most public APIs are compatible with mysqlclient and MySQLdb. PyMySQL works with MySQL 5.5+ and MariaDB 5.5+.

MySQL is a leading open source database management system. It is a multiuser, multithreaded database management system. MySQL is especially popular on the web.

cities\_mysql.sql

USE testdb;

DROP TABLE IF EXISTS cities;

CREATE TABLE cities(id INT PRIMARY KEY AUTO\_INCREMENT, name VARCHAR(255), population INT);

INSERT INTO cities(name, population) VALUES('Bratislava', 432000);

INSERT INTO cities(name, population) VALUES('Budapest', 1759000);

INSERT INTO cities(name, population) VALUES('Prague', 1280000);

INSERT INTO cities(name, population) VALUES('Warsaw', 1748000);

INSERT INTO cities(name, population) VALUES('Los Angeles', 3971000);

INSERT INTO cities(name, population) VALUES('New York', 8550000);

INSERT INTO cities(name, population) VALUES('Edinburgh', 464000);

INSERT INTO cities(name, population) VALUES('Berlin', 3671000);

In the tutorial, we use the cities table.

PyMySQL installation

$ sudo pip3 install pymysql

We use the pip3 tool to install PyMySQL.

PyMySQL version example

In the following example, we get the version of MySQL.

version.py

#!/usr/bin/python

import pymysql

con = pymysql.connect('localhost', 'user7',

's$cret', 'testdb')

try:

with con.cursor() as cur:

cur.execute('SELECT VERSION()')

version = cur.fetchone()

print(f'Database version: {version[0]}')

finally:

con.close()

In MySQL, we can use SELECT VERSION() to get the version of MySQL.

import pymysql

We import the pymysql module.

con = pymysql.connect('localhost', 'user7',

's$cret', 'testdb')

We connect to the database with connect. We pass four parameters: the hostname, the MySQL user name, the password, and the database name.

with con.cursor() as cur:

Using the with keyword, the Python interpreter automatically releases the resources. It also provides error handling. We get a cursor object, which is used to traverse records from the result set.

cur.execute('SELECT VERSION()')

We call the execute function of the cursor and execute the SQL statement.

version = cur.fetchone()

The fetchone function fetches the next row of a query result set, returning a single sequence, or None when no more data is available.

print(f'Database version: {version[0]}')

We print the version of the database.

finally:

con.close()

The pymysql module does not implement the automatic handling of the connection resource; we need to explicitly close the connection with close in the finally clause.

## B3.How does Python integrate with SQL?

Answer:

Here, we are going to connect SQLite with Python. Python has a native library for SQLite. Let us explain how it works.

To use SQLite, we must import sqlite3.

Then create a connection using connect() method and pass the name of the database you want to access if there is a file with that name, it will open that file. Otherwise, Python will create a file with the given name.

After this, a cursor object is called to be capable to send commands to the SQL. Cursor is a control structure used to traverse and fetch the records of the database. Cursor has a major role in working with Python. All the commands will be executed using cursor object only.

To create a table in the database, create an object and write the SQL command in it with being commented. Example:- sql\_comm = ”SQL statement”

And executing the command is very easy. Call the cursor method execute and pass the name of the sql command as a parameter in it. Save a number of commands as the sql\_comm and execute them. After you perform all your activities, save the changes in the file by committing those changes and then lose the connection.

filter\_none

edit

play\_arrow

brightness\_4

# Python code to demonstrate table creation and

# insertions with SQL

# importing module

import sqlite3

# connecting to the database

connection = sqlite3.connect("myTable.db")

# cursor

crsr = connection.cursor()

# SQL command to create a table in the database

sql\_command = """CREATE TABLE emp (

staff\_number INTEGER PRIMARY KEY,

fname VARCHAR(20),

lname VARCHAR(30),

gender CHAR(1),

joining DATE);"""

# execute the statement

crsr.execute(sql\_command)

# SQL command to insert the data in the table

sql\_command = """INSERT INTO emp VALUES (23, "Rishabh", "Bansal", "M", "2014-03-28");"""

crsr.execute(sql\_command)

# another SQL command to insert the data in the table

sql\_command = """INSERT INTO emp VALUES (1, "Bill", "Gates", "M", "1980-10-28");"""

crsr.execute(sql\_command)

# To save the changes in the files. Never skip this.

# If we skip this, nothing will be saved in the database.

connection.commit()

# close the connection

connection.close()

## B4.What is MySQLdb Python?What is Generic database interface?

Answer:

MySQLdb is an application programming interface which enables you to connect a Python program with a database server, and it is built on top MySQL C API.

To perform the connection, open Python Shell, and type the following command

1

>> import MySQLdb

If the above statement executes successfully, then you can go ahead with creating a database connection, and executing SQL queries.

The import statement may give the following error

1

2

3

4

5

>> import MySQLdb

Traceback (most recent call last):

File "<pyshell#1>", line 1, in

import MySQLdb

ModuleNotFoundError: No module named 'MySQLdb'

Python DB – API module is not installed on your machine if you get this error. You can download the MySQL database connector for Python from SourceForge.

Alternatively, you can also download and install Python MySQL connector from their official website – MySQL/Python Connector.

Once you’ve successfully established the Python database connection with the MySQL server, you can now start writing the following Python code to check the database connectivity and perform the basic CRUD operations on the database.

Python Database Connection To MySQL Program

Here’s a sample code to check the version of the MySQL database server that the Python program connects to.

1

2

3

4

5

6

7

8

import MySQLdb

conn = MySQLdb.connect("localhost", "username", "password","sample\_db")

cursor = conn.cursor()

cursor.execute("SELECT VERSION()")

x = cursor.fetchone()

print "The MySQL database server version is:", x[0]

cursor.close()

conn.close()

This is how we can connect to the MySQL server from a Python application. Let us now see the different functions available to perform the CRUD operations.

## I1.what is relational database system interface?

Answer:

The RDBMS provides an interface between users and applications and the database, as well as administrative functions for managing data storage, access, and performance.

Several factors can guide your decision when choosing among database types and relational database products. The RDBMS you choose will depend on your business needs. Ask yourself the following questions:

What are our data accuracy requirements? Will data storage and accuracy rely on business logic? Does our data have stringent requirements for accuracy (for example, financial data and government reports)?

Do we need scalability? What is the scale of the data to be managed, and what is its anticipated growth? Will the database model need to support mirrored database copies (as separate instances) for scalability? If so, can it maintain data consistency across those instances?

How important is concurrency? Will multiple users and applications need simultaneous data access? Does the database software support concurrency while protecting the data?

What are our performance and reliability needs? Do we need a high-performance, high-reliability product? What are the requirements for query-response performance? What are the vendor’s commitments for service level agreements (SLAs) or unplanned downtime?

The Relational Database of the Future: The Self-Driving Database

Find out more about how self-driving databases work (14:32)

Over the years, relational databases have gotten better, faster, stronger, and easier to work with. But they’ve also gotten more complex, and administering the database has long been a full-time job. Instead of using their expertise to focus on developing innovative applications that bring value to the business, developers have had to spend most of their time on the management activity needed to optimize database performance.

Today, autonomous technology is building upon the strengths of the relational model to deliver a new type of relational database. The self-driving database (also known as the autonomous database) maintains the power and advantages of the relational model but uses artificial intelligence (AI), machine learning, and automation to monitor and improve query performance and management tasks. For example, to improve query performance, the self-driving database can hypothesize and test indexes to make queries faster, and then push the best ones into production—all on its own. The self-driving database makes these improvements continuously, without the need for human involvement.

Autonomous technology frees up developers from the mundane tasks of managing the database. For instance, they no longer have to determine infrastructure requirements in advance. Instead, with a self-driving database, they can add storage and compute resources as needed to support database growth. With just a few steps, developers can easily create an autonomous relational database, accelerating the time for application development.