Hyperiondev

TASK

SQLite

Visit our website

Introduction

WELCOME TO THE SQLITE TASK!

In this task, learn how to write code to create and manipulate a database with SQLite.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

INTRODUCTION TO SQLITE

SQLite is built in Python to provide a simple relational database management system (RDBMS). It is very easy to set up, very fast and lightweight, and thus referred to as 'lite'. Important features to note about SQLite are that it is self-contained, serverless, and transactional and requires zero-configuration to run.

- **Self-contained:** this means that it does not need much support from the operating system or external libraries. This makes it suited for use in embedded devices like mobile phones, iPods and game devices that lack the infrastructure of a computer. The source code is found in files called sqlite3.c and the header file sqlite3.h. When you want to use SQLite in an application, ensure that you have these files in your project directory when compiling your code.

- **Serverless:** in most cases, RDBMSs require a separate server to receive and respond to requests sent from the client.

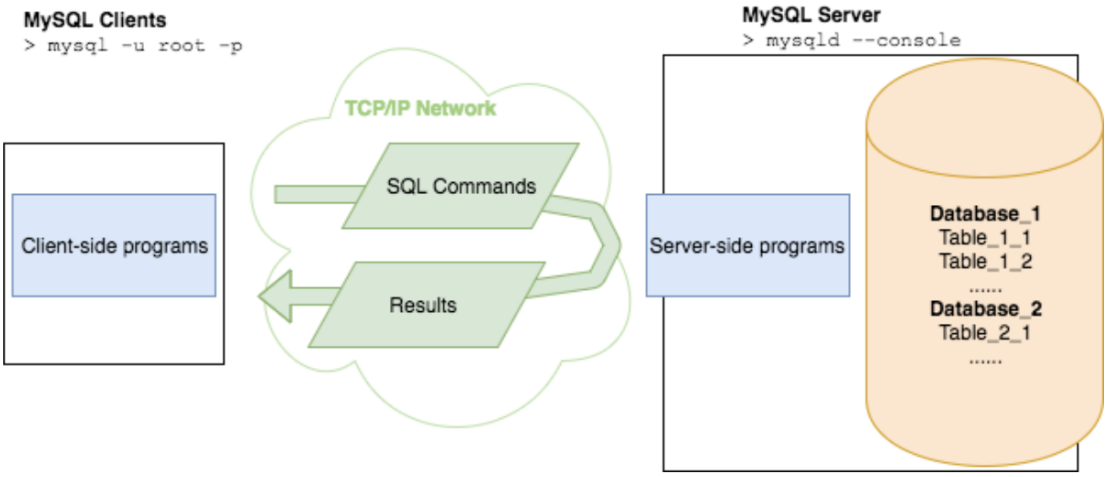

Image source: ***SQLite Is Severless***

Such systems include MySQL and the Java Database Client– JDBC. These clients have to use the TCP/IP protocol to send and receive responses. This is referred to as the Client/Server architecture. SQLite does not make use of a separate server. While using SQLite, the application reads and writes directly to the database files stored on the application server disk.

- **Transactional:** all transactions in SQLite are atomic, consistent, isolated, and durable (**ACID**-compliant). In other words, if a transaction occurs that attempts to make changes to the databases, the changes will either be made in all the appropriate places (in all linked tables and affected rows) or not at all. This is to ensure data integrity.

- **Zero-configuration required:** you don't need to install SQLite prior to using it in an application or system. This is because of the serverless characteristic described previously.

PYTHON'S SQLITE MODULE

It is easy to create and manipulate databases with Python. To allow us to use SQLite with Python, the Python Standard Library includes a module called "sqlite3". To use the SQLite3 module, we need to add an import statement to our Python script:

```
import sqlite3
```

We can then use the function `sqlite3.connect` to connect to the database. We pass the name of the database file to open or create it.

```
# Creates or opens a file called student_db with a SQLite3 DB

db = sqlite3.connect('data/student_db')
```

Creating and Deleting Tables

To make any changes to the database, we need a **cursor object**. A cursor object is an object that is used to execute SQL statements. Next, `.commit` is used to save changes to the database. It is important to remember to commit changes since this ensures the atomicity of the database. If you close the connection using `close` or the connection to the file is lost, changes that have not been committed will be lost.

Below we create a student table with id, name, and grade columns.

```
cursor = db.cursor()  # Get a cursor object

cursor.execute('''
    CREATE TABLE student(id INTEGER PRIMARY KEY, name TEXT,
```

```
                    grade INTEGER)
''')
db.commit()
```

Always remember that the **commit** function is invoked on the **db** object, not the **cursor** object. If we type **cursor.commit,** we will get the following error message: “AttributeError: 'sqlite3.Cursor' object has no attribute 'commit'”.

Inserting into the Database

To insert data, we use the cursor again to execute a SQL statement. When using data stored in Python variables to insert data into a database table, use the "?" placeholder. It is not secure to use string operations or concatenation to make your queries.

In this example, we are going to insert two students into the database; their information is stored in Python variables.

```
name1 = 'Andres'
grade1 = 60


name2 = 'John'
grade2 = 90

# Insert student 1
cursor.execute('''INSERT INTO student(name, grade)
                  VALUES(?,?)''', (name1,grade1))
print('First user inserted')

# Insert student 2
cursor.execute('''INSERT INTO student(name, grade)
                  VALUES(?,?)''', (name2,grade2))
print('Second user inserted')

db.commit()
```

In the example above, the values of the Python variables are passed inside a **tuple**. You could also use a dictionary with the named style placeholder:

```
name3 = 'Sheila'
grade3 = 40
```

```
cursor.execute('''INSERT INTO student(name, grade)
                  VALUES(:name,:grade)''',
                  {'name':name3, 'grade':grade3})
```

If you need to insert several users, use **executemany** and a list with the tuples:

```
students_ = [(name1,grade1),(name2,grade2),(name3,grade3)]

cursor.executemany(''' INSERT INTO student(name, grade) VALUES(?,?)''',
students_)

db.commit()
```

If you need to get the ID of the row you just inserted, use **lastrowid**:

```
id = cursor.lastrowid
print('Last row id: %d' % id)
```

Use **rollback** to roll back any change to the database since the last call to commit:

```
cursor.execute('''UPDATE student SET grade = ? WHERE id = ? ''', (65, 2))

db.rollback()
```

Retrieving Data

To retrieve data, execute a **SELECT** SQL statement against the cursor object and then use **fetchone()** to retrieve a single row or **fetchall()** to retrieve all the rows.

```
id = 3
cursor.execute('''SELECT name, grade FROM student  WHERE id=?''', (id,))
student = cursor.fetchone()

print(student)
```

The cursor object works as an iterator, invoking **fetchall()** automatically:

```
cursor.execute('''SELECT name, grade FROM student''')
for row in cursor:
    # row[0] returns the first column in the query (name), row[1] returns
the email column.
```

```
    print('{0} : {1}'.format(row[0], row[1]))
```

Updating and Deleting Data

Updating or deleting data is similar to inserting data:

```
 # Update user with id 1
grade = 100
id = 1
cursor.execute('''UPDATE student SET grade = ? WHERE id = ? ''', (grade,
id))

# Delete user with id 2
id = 2
cursor.execute('''DELETE FROM student WHERE id = ? ''', (id,))

cursor.execute('''DROP TABLE student''')

db.commit()
```

When we are done working with the DB, we need to close the connection:

```
 db.close()
```

SQLite Database Exceptions

It is very common for exceptions to occur when working with databases. It is important to handle these exceptions in your code.

In the example below, we use a try/except/finally clause to catch any exception in the code. We put the code that we would like to execute but that may throw an exception (or cause an error) in the **`try`** block. Within the **`except`** block, we write the code that will be executed if an exception does occur. If no exception is thrown, the except block will be ignored. The **`finally`** clause will always be executed, whether an exception was thrown or not. When working with databases, the **`finally`** clause is very important, because it always closes the database connection correctly. Find out more about exceptions **here**.

```
try:
    # Creates or opens a file called student_db with a SQLite3 DB
```

```
    db = sqlite3.connect('student_db')
    # Get a cursor object
    cursor = db.cursor()
    # Checks if the table “users” exists and if not creates it
    cursor.execute('''CREATE TABLE IF NOT EXISTS
                      users(id INTEGER PRIMARY KEY, name TEXT, grade
INTEGER)''')
    # Commit the change
    db.commit()
# Catch the exception
except Exception as e:
    # Roll back any change if something goes wrong
    db.rollback()
    raise e
finally:
    # Close the db connection
    db.close()
```

Notice that the except block of our try/except/finally clause in the example above will be executed if any type of error occurs:

```
 # Catch the exception
except Exception as e:
    raise e
```

This is called a catch-all clause. In a real application, you should catch a specific exception. To see what type of exceptions could occur, see **DB-API 2.0 Exceptions**.

Instructions

First, read **example.py**; open it using VS Code or a similar editor.

- **example.py** should help you understand some simple Python.
- You may run example.py to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with Python.

Compulsory Task 1

Follow these steps:

- Create a python file called **database_manip.py**. Write the code to do the following tasks:
    - Create a table called python_programming.
    - Insert the following new rows into the python_programming table:

id	name	grade
55	Carl Davis	61
66	Dennis Fredrickson	88
77	Jane Richards	78
12	Peyton Sawyer	45
2	Lucas Brooke	99

    - Select all records with a grade between 60 and 80.
    - Change Carl Davis's grade to 65.
    - Delete Dennis Fredrickson's row.
    - Change the grade of all people with an id below than 55.

Completed the task(s)?

Ask an expert to review your work!

Review work

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved? Do you think we've done a good job?

Click here to share your thoughts anonymously.

REFERENCES:

DomainsatCost.com. (2015, November 2). MySQL, Database and SQL Defined. Retrieved April 15, 2019, from **http://yourdomaingoeshere.com/mysql-database-and-sql-defined/**

Python Central. (2013, April 11). Introduction to SQLite in Python. Retrieved April 15, 2019, from Python Central: **https://www.pythoncentral.io/introduction-to-sqlite-in-python/**