Creating and Using an SQLite Database

# Using a Database for Your Project

One of the technical skills that you can gain marks for on your project is the use of a database.

## Database tools:

**SQLite** – a widely used database. Unlike most databases it is stored as a file on your computer, rather than being accessed via the internet. However, this should be suitable for your project.

**sqlite3 module** – this is a Python module that allows you to build and use an SQLite Database via Python commands

**PyCharm -**  PyCharm has got some great inbuilt tools for viewing and manipulating an SQLite database

# Creating an SQLite database

Connect to an existing SQLite database in your project by importing the sqlite3 command and using the sqlite3.connect command. If you refer to a database that is not in your file path this command will create a new database.

Interactions with the database always start with creating a connection to the database and a *cursor*. The cursor is a pointer to a specific row in your database and will be the point at which the database is written to or read from.

Create a new module *create\_student\_database.py* and use the commands below.

Create a database connection and cursor using these commands:

import sqlite3

conn = sqlite3.connect("student.sqlite")

cursor = conn.cursor()

If you leave connections to the database open it can take up memory and slow down the database, so when you finish writing commands you should close the connection:

conn.close()

## Create an initial table

To interact with the database we write commands in SQL (Structured Query Language). We can write these commands at Python text strings and then execute them using the cursor.

Here is an SQL string that will create a *students* table, and the command to execute the string:

create\_students\_table = """  
CREATE TABLE IF NOT EXISTS students (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 firstname TEXT NOT NULL,  
 lastname TEXT NOT NULL,  
 age INTEGER,  
 gender TEXT  
);  
"""  
cursor.execute(create\_students\_table)

The whole code is here:

import sqlite3  
  
*# Create a connection to a database - if one does not exist, a new one will be created*conn = sqlite3.connect("student.sqlite")  
  
*# A cursor is a pointer to a place in the database which allows access  
# to a table row-by-row*cursor = conn.cursor()  
  
*# SQL commands can be written as text and then 'run' using an execute command  
# Example - creating a table*create\_students\_table = """  
CREATE TABLE IF NOT EXISTS students (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 firstname TEXT NOT NULL,  
 lastname TEXT NOT NULL,  
 age INTEGER,  
 gender TEXT  
);  
"""  
cursor.execute(create\_students\_table)  
  
conn.close()

If you run the code, you will see that it creates a file called “student.sqlite” in your project directory. Double-click on this and you will be able to add it to the PyCharm database window, which is a useful tool for viewing and manipulating the database.

Refresh the student\_db in the Database window and double-click on the students table to see the (so far empty) table.

Questions

1. What columns have been generated in the students table?
2. What does Primary Key mean?
3. What does AUTOINCREMENT mean?

# Writing data to the database

SQL commands are also used to write data to the database. The method is similar to above:

* Create a database connection
* Create a cursor
* Write an INSERT command as a string
* Execute the command.

However, before any changes are written to the database, they must also be committed.

For example, to write a single student into the students table use, create a new python module called *write\_to\_student\_db.py* and type in the following code:

### Writing a single record

import sqlite3

conn = sqlite3.connect("student.sqlite")

cursor = conn.cursor()

insert\_query = """  
INSERT INTO   
 students (firstname, lastname, age, gender)  
VALUES  
 ('Hermione', 'Grainger', 14, 'Female');  
"""

cursor.execute(insert\_query)  
conn.commit()

conn.close()

Questions

1. Why does the database allow you to queue up a set of changes before committing them all to be written?
2. Why do we not need to include an id in the values that we insert?

## Parameterized Queries

To write several lines of information to the database it is useful to use a parameterized query. The query is set up with placeholders for the values.

The query can then be executed repeatedly using a tuple of values to replace the placeholders.

E.g.

parameterised\_insert\_query = """  
INSERT INTO   
 students (firstname, lastname, age, gender)  
VALUES  
 (?, ?, ?, ?);  
"""

cursor.execute(parameterised\_insert\_query, ("Harry", "Potter", 13, "male"))

conn.commit()

conn.close()

will add Harry Potter to the database.

Of course this works better with repeated calls to cursor.execute. In the example below, we will use the faker module to insert a set of random names into the database.

You will need to install the faker module. You can include the latest version: faker~=22.2.0 in your requirements.txt file.

### Write some fake names to the database

Change your *write\_to\_student\_db* to the following:

import sqlite3  
from faker import Faker  
import random  
  
fake = Faker('en\_GB')  
  
parameterised\_insert\_query = """  
INSERT INTO   
 students (firstname, lastname, age, gender)  
VALUES  
 (?, ?, ?, ?);  
"""  
  
conn = sqlite3.connect("student.sqlite")  
cursor = conn.cursor()  
  
fake.random.seed(4321)  
random.seed(4321)  
for \_ in range(10):  
 f\_name = fake.first\_name()  
 l\_name = fake.last\_name()  
 age = random.randint(11, 18)  
 gender = random.choice(('male', 'female'))  
 cursor.execute(parameterised\_insert\_query,  
 (f\_name, l\_name, age, gender))  
  
conn.commit()  
conn.close()

Questions

1. What do random.seed() and fake.random.seed do?

## Execute many SQL commands with a single statement

The sqlite3 module includes the .executemany() method on the cursor object to do this a bit quicker:

data = [(fake.first\_name(),   
 fake.last\_name(),   
 random.randint(11, 18),   
 random.choice(('male', 'female')))  
 for \_ in range(20)]  
cursor.executemany(parameterised\_insert\_query, data)  
conn.commit()

conn.close()

## Updating Records

Records can also be updated using update\_queries:

*# Updating a record*update\_query = """  
UPDATE students  
SET lastname = ?  
WHERE id = 4;  
"""  
cursor.execute(update\_query, ('Smith',))  
conn.commit()  
  
  
*# Updating lots of records - increase all the ages by 1*increment\_age\_query = """  
UPDATE students  
SET age = age + 1;  
"""  
cursor.execute(increment\_age\_query)  
conn.commit()

conn.close()

The whole code is here:

*# write\_to\_student\_db.py*import sqlite3  
from faker import Faker  
import random  
  
*# Create a connection to the database*conn = sqlite3.connect("student.sqlite")  
  
*# Create a cursor*cursor = conn.cursor()  
  
*# Example - adding a user to the students table*insert\_query = """  
INSERT INTO   
 students (firstname, lastname, age, gender)  
VALUES  
 ('Hermione', 'Grainger', 14, 'female');  
"""  
cursor.execute(insert\_query)  
conn.commit()  
  
*# Example - parameterized query*parameterised\_insert\_query = """  
INSERT INTO   
 students (firstname, lastname, age, gender)  
VALUES  
 (?, ?, ?, ?);  
"""  
cursor.execute(parameterised\_insert\_query, ("Harry", "Potter", 13, "male"))  
  
*# The Faker module gives a way of creating random data*fake = Faker('en\_GB')  
random.seed(4321)  
fake.random.seed(4321)  
for \_ in range(10):  
 f\_name = fake.first\_name()  
 l\_name = fake.last\_name()  
 age = random.randint(11, 18)  
 gender = random.choice(('male', 'female'))  
 cursor.execute(parameterised\_insert\_query,  
 (f\_name, l\_name, age, gender))  
conn.commit()  
  
*# Another way of doing a parameterized query*data = [(fake.first\_name(),  
 fake.last\_name(),  
 random.randint(11, 18),  
 random.choice(('male', 'female')))  
 for \_ in range(20)]  
cursor.executemany(parameterised\_insert\_query, data)  
conn.commit()  
  
*# Updating a record*update\_query = """  
UPDATE students  
SET lastname = ?  
WHERE id = 4;  
"""  
cursor.execute(update\_query, ('Smith',))  
conn.commit()

conn.close()

## Using the database tools

Now that you have a table with data in, you can explore the PyCharm database tools.

* Click on the *students* table to view and filter the student data
* Use the console to write SQL statements directly. For example, can you write and SQL statement to set the first name of student with id 13 to “Jeffrey”?

# Reading data from the database

Create a new module *read\_from\_student\_db.py*

Reading data is done via the SELECT query. The process is similar to before, with a different final stage:

* Create a database connection
* Create a cursor
* Write a SELECT command as a string
* Execute the command.
* Fetch one, several or all of the results
* Remember to close the connection

### Reading data from the students table

import sqlite3  
conn = sqlite3.connect("student.sqlite")  
cursor = conn.cursor()  
  
select\_students = """  
SELECT id, firstname, lastname  
FROM students  
WHERE age >= 15  
"""  
  
cursor.execute(select\_students)  
first\_student = cursor.fetchone()  
more\_students = cursor.fetchmany(10)  
other\_students = cursor.fetchall()

conn.close()

Note that the cursor moves through the database, so more\_students and other\_students will not include the students fetched by the previous commands.

## Aggregate Query

We can also write an SQL query to aggregate data according to a criteria.

### Find the average age

average\_query = """  
SELECT avg(age)  
FROM students  
WHERE gender = ?  
"""  
average\_age = cursor.execute(average\_query, ('female',)).fetchone()[0]

## Aggregate Group by Query

group\_by\_query = """  
SELECT gender, avg(age)  
FROM students  
GROUP BY gender  
"""  
average\_age\_by\_gender = cursor.execute(group\_by\_query).fetchall()

Exercises

1. Write and execute a SELECT query to fetch up to 5 students with a firstname starting with ‘J’
2. Write, execute and fetch the number of students grouped by gender
3. Write, execute and fetch the sum of the student’s ages grouped by the first letter of their firstnames. (Hint: use substr to find the first letter).

## Code Repository

This example code can be found in the GitHub Repository: <https://github.com/AndrewDales/database_examples>