Python and Databases

Wednesday 25th March
CAS North East Conference, Newcastle

Sue Sentance
King's College London/CAS/Python School
@suesentance
sue.sentance@computingatschool.org.uk

This handout includes notes and snippets of code to help you with today's session

The objectives are to:

- Understand how to create a database and table in Python
- Learn how to add, delete and update records
- Learn how to search for and sort data using one table
- Perhaps put all this together to create a menu-based system

Databases recap

In this course we assume that you have a basic grounding in how databases work and are familiar with these terms:

- Table
- Field
- Record
- Primary Key

The database we are using in this session is a simple, one-table database of films. Here is the data dictionary for this table

Field name	**Data type**	**Other information**
FilmID	Integer	Primary Key
Title	String	
Genre	String	
Year	Integer	

We will not be using relationships between tables in this session (not enough time!) but when you progress to this you will need to understand the following concepts:

- Foreign Key
- Relationships (one-to-one, one-to-many, many-to-many)

SQL – the language of databases

SQL is a standard language for communicating with databases. It is completely independent of programming language or implementation. It is not case-sensitive. Understanding SQL can help you with any database work you do, regardless of whether you are using Python or not.

There are a few key SQL commands that you need to know which are:

- Create Table
- Select … From … Where
- Insert Into
- Delete
- Update

Below are some examples using the Film table.

To create a table

```
CREATE TABLE Film(
    FilmID integer,
    Title text,
    Genre real,
    Year integer
    Primary Key(FilmID));
```

Adding records in SQL

```
INSERT INTO Film (Title, Genre, Year)
VALUES ('Boyhood','Family',2014)
```

Querying the database in SQL

```
SELECT * FROM Film
```

Updating data

```
UPDATE Film
SET Genre=?, Year=?
WHERE FilmID=?
```

Deleting data

```
DELETE FROM Film
WHERE FilmID=?
```

Searching for data

```
SELECT *
FROM Film
WHERE Year > 2000
```

Python and Databases

sqlite3 comes free with Python and enables you to create and work with databases. An sqlite3 database is stored locally on your machine.

When working with Python and databases, there are a few key commands that you will use a lot. These involve opening the database, sending it commands and saving changes. Inside these commands we need to put the SQL statements that we will send to the database.

We use the with statement when opening the database as this ensures that if the program crashes, the database is closed and not left in an open state (thus preventing errors).

Key commands for working with Python and databases are:

`sqlite.connect(<database name>)`	Forms the connection with the database
`cursor = db.cursor()`	This identifies the cursor for the database which tells Python where in the database you are looking at the time
`cursor.execute(<sql statements>)`	This is used to pass all sql statements to the database
`db.commit()`	This saves the changes made

Data types in sqlite3

Each value stored in an SQLite database (or manipulated by the database engine) has one of the following storage classes:

NULL. The value is a NULL value.

INTEGER. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.

REAL. The value is a floating point value, stored as an 8-byte IEEE floating point number.

TEXT. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).

BLOB. The value is a blob of data, stored exactly as it was input.

This is only a subset of the data types used by SQL in general. There are many more so if you progress to using MySQL (for example, to create a server-side database) you will need to be familiar with others. If you use other data types with sqlite they will always map to the five given here.

Creating a table

To create a database in Python from scratch just enter the following

```
import sqlite3

with sqlite3.connect('film.db') as db:
    pass
```

To create a table, in this case a Film table we need to use the Create Table command.

As above, the SQL is

```
CREATE TABLE Film(
    FilmID integer,
    Title text,
    Genre real,
    Year integer
    Primary Key(FilmID));
```

We need to put this inside a cursor.execute() statement for it to be carried out by Python.

Then a `db.commit()` statement is needed to save the data.

```
import sqlite3

def create_film_table():
    with sqlite3.connect("film.db") as db:
        cursor = db.cursor()
        # the cursor is essential to navigate around the database
        cursor.execute("""
            CREATE TABLE Film(
            FilmID integer,
            Title text,
            Genre real,
            Year integer,
            Primary Key(FilmID));""")
        db.commit()
```

Adding records

To insert a record into the database use the following SQL command

```
INSERT INTO Film (Title, Genre, Year)
VALUES ('Wild','Drama',2014)
```

Notice that you do not need to provide a values for **FilmID** - this is generated by SQLite automatically.

Wrapped up inside a cursor.execute() command this would give you:

```
def insert_records():
    with sqlite3.connect("film.db") as db:
        cursor = db.cursor()
        # the cursor is essential to navigate around the database
        sql = """insert into Film (Title, Genre, Year)
               values("Wild","Drama",2014)"""
        cursor.execute(sql)
        db.commit()
```

Don't forget to use db.commit() as this saves the data.

This works fine, but gets a bit cumbersome as the sql statement is hard-coded. We can use "?" in the sql statement to represent data we do not know yet – thus creating a parameter query.

```
def insert_records(values):
    with sqlite3.connect("film.db") as db:
        cursor = db.cursor()
        # the cursor is essential to navigate around the database
        sql = """insert into Film (Title, Genre, Year)
                 values(?,?,?)"""
        cursor.execute(sql,values)
        db.commit()

next_film = ("Wild","Drama",2014)
insert_records(next_film)
```

? is inserted instead of actual values.

Pass in the values to the cursor.execute command

Values are passed in to the function through a parameter

Task

Using the second method, add some more films to the table, for example:

Title	Genre	Year
The Lion King	Family	1994
Django Unchained	Western	2012
Selma	Drama	2014
Boyhood	Family	2014
Gone Girl	Drama	2014
…..		
……		

Extension: Write a program which asks the user to enter the Title, Genre and Year of a film then enter it into the table.

Querying data

Before going any further, it would be good to find out what is in the database by querying it.

The SQL Statement to select all data from the film table is

```
SELECT * FROM Film
```

To select certain films we use WHERE in our statement. Here we want just the films that were made in 2014

```
SELEECT * FROM Film WHERE year = 2014
```

To only show certain films we need to put the fields we need instead of the wildcard *

```
SELECT (Title, Year) from Film
```

To show films in order we use the statement ORDER BY

```
SELECT (Title, Year) from Film ORDER BY Year Asc
```

To write this in Python, again, you need to put the sql statement inside a cursor.execute() statement. You can now use these very useful commands:

`cursor.fetchall()` – returns all records that were selected by the SELECT command

`cursor.fetchone()` – returns one record that was selected by the SELECT command (only use this when you only expect one.

The Python code for selecting all records is:

```
def select_all_films():
    with sqlite3.connect("film.db") as db:
        cursor = db.cursor()
        # the cursor is essential to navigate around the database
        sql= "Select * from Film"
        cursor.execute(sql)
        result = cursor.fetchall()
        print(result)
```

If only one record is needed, for example, selecting a record by ID, use this code:

```
def select_one_film():
    with sqlite3.connect("film.db") as db:
        cursor = db.cursor()
        # the cursor is essential to navigate around the database
        sql= "Select * from Film where FilmID = 1"
        cursor.execute(sql)
        result = cursor.fetchone()
        print(result)

select_one_film()
```

This is a little clunky as ideally we would want to use the return statement to return the films from the function and then print them out in the main program. We will do that next.

In addition is useful to pass into the function what you would like to find. The next program gives an example of asking the user for a particular genre and then printing out the films for that genre.

```
def select_genre(genre):
    with sqlite3.connect("film.db") as db:
        cursor = db.cursor()
        sql= "Select * from Film where genre = ?"
        cursor.execute(sql,(genre,))
        result = cursor.fetchall()
        return result


choose_genre = input("What genre would you like to see?")
films = select_genre(choose_genre)
print(films)
```

The output from a `fetchall()` is a list of tuples. There are ways of making the output more presentable that I will demonstrate in the session.

Task

Write some programs which can:

- Find all films in your table since 2000
- Order films by year (see the SQL examples at the top of the handout)
- Find all films which were made in 2014 and are Drama genre.

Consolidation task

Create a menu-based system that allows you to add records, update films, delete films and find particular types of films. The menu for your system could resemble the following screenshot (but you can choose your own menu items if you wish):

```
Film database
=============

Please select an option

1 - Add a new film
2 - Delete a film
3 - Show all films
4 - Find films by year
5 - Exit

>>>
```

Further topics to look at

This has just been an introduction to this topic but you can continue learning!

Use http://pythonschool.net to continue learning about this topic. This site has video tutorials to show you how to develop more sophisticated databases, and an associated github site with answers to the exercises.

In the databases section of Python School you learn how to:

- Create a Coffee Shop database
- Use a tool called sqlite inspector to examine your tables
- Create relationships between one or more tables
- Write query statements using more than one table

In the Server-side scripting section of Python School you can also learn how to create online databases using server-side scripting and the more powerful database engine MySQL.

Thanks for attending!

Please complete the evaluation form for this session at:
https://www.surveymonkey.com/s/NOEFirstFeedback

(under the Master Teacher code, choose “University Partner” and then select “King’s College London”).

Sue Sentance (sue.sentance@kcl.ac.uk)