Lecture: Python and Databases

Objectives of lecture

- Create a database and table using Python and SQLite
- Add records
- Search for data
- Delete records
- Update records

Database terms

- Table - collection of related data held in a table format within a database.
- Database tables consist of rows and columns.
- Primary Key - used to uniquely identify a record in a table

SQL – the language of databases

- SQL is a standard language for communicating with databases by carrying out queries.
- SQL stands for Structured Query Language.
- SQL can be pronounced as both *sequel* or S-Q-L.
- It is independent of the programming language we are using (in this case Python).
- It is **not** case-sensitive.

SQLite

- SQLite can be used with many programming languages.
- SQLite does NOT require a server to run (server-less).
- Provides a **local** database to store data.
- **sqlite3** comes with Python.
- There are key SQL commands that you need to know:
    - Create Table
    - Select … From … Where
    - Insert Into
    - Update
    - Delete

SQLite Database with Python

Key commands for working with Python and databases are shown in the table.

Key commands	**Description**
`db = sqlite3.connect(<database name>)`	Forms the connection with the database. If the file exists, it opens it. If the file does not exist, it creates it.
`cursor = db.cursor()`	To execute SQLite statements in Python, you need a **cursor** object created by using the *cursor()* method.
`cursor.execute(<sql statements>)`	Used to execute a SQL statement.
`db.commit()`	Save (commit) the changes permanently.
`db.close()`	Closes the database connection. Need to commit() first or you changes will be lost.

SQLite and Python types

SQLite supports the following types: NULL, INTEGER, REAL, TEXT, BLOB. The following table shows how the SQLite types map to the corresponding Python types.

SQLite type	Python type
NULL	None
INTEGER	int
REAL	float
TEXT	str
BLOB	bytes

Creating the database

We will use Python and SQLite to create a database *film.db* to hold film information. The first steps are as follows.

1. Import the sqlite3 library.
2. Make a connection by creating a *Connection* object that will represent the database. This object is created using SQLite's connect() function.
3. Create a cursor object. A cursor object allows us to execute SQL queries against a database.

```
import sqlite3

db = sqlite3.connect("film.db")
```

SQLite3 Cursor Object

To execute SQLite statements in Python, you need a **cursor** object. You can create it using the *cursor()* method. We will create a variable *cursor* to hold our cursor object:

```
cursor = db.cursor()
```

We can use the cursor object to call the execute() method to perform any SQL queries - cursor.execute()

Creating a table

The *film.db* database will have the one table called *Film* with the following fields.

Table *Film*		
Field name	**Data type**	**Other information**
FilmID	Integer	Primary Key
Title	Text	
Genre	Text	
Year	Integer	

To create the ***Films*** table we use the SQL Create Table command.

Then we use the cursor.execute() statement for the SQL to be carried out.

Example: Creating the *film* table

```
import sqlite3                          # Import sqlite3 library


db = sqlite3.connect("film.db")         # Create Connection object
cursor = db.cursor()                    # Create Cursor object

# Create a string to hold the SQL query
sql = """
    CREATE TABLE IF NOT EXISTS Film
    (FilmID integer PRIMARY KEY,
    Title text,
    Genre text,
    Year integer);"""

cursor.execute(sql)                     # Pass string variable
db.commit()                             # Save permanently
```

Notes on example:

1. Here we pass a string containing the SQL query including the CREATE TABLE statement to the `execute()` method of the `Cursor` object.
   - Wrap the SQL query in quotes. Use single, double, or triple quotes. However, triple quotes allow you to write multi-line queries.
2. We create four columns: *FilmID*, *Title*, *Genre, Year.*
3. *FilmID* is assigned as the primary key.
4. Note: If we try to create a table that already exists it will produce an error. To check if the table doesn't already exist, we use IF NOT EXISTS with the CREATE TABLE statement.
5. Commit the changes (save permanently) by using the `commit()` function on the `Connection` object.

Adding records

To insert a record into the database we use the following SQL syntax:

```
INSERT INTO table (column1,column2 ,..) VALUES ( value1, value2 ,...);
```

- Specify the name of the table we want to add values into.
- Specify a list of all the columns in the table. While this list is optional, it's good practice to include it. Then follow with the list of values to include.
- If we don't include all the column names, we have to include a value for each column.

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

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

- Reminder - wrap SQL query in quotes. It doesn't matter if we use single, double, or triple quotes. Triple quotes allow you to write multi-line queries.
- This example does not include a value for the FilmID. The FilmID column was defined as "integer PRIMARY KEY". The values for this will be generated by SQLite automatically.
- To insert rows we use the cursor object to execute the query.
- Note: recommended to use single quotes for literal strings within the SQL statements.

Method 1

- Create a string containing the SQL statement and pass that string to the cursor.execute() command:

```
sql = '''INSERT INTO Film (Title, Genre, Year)  VALUES ('Wild','Drama',2014); '''
cursor.execute(sql)
```

Method 2

- Pass the data directly to the cursor.execute() command:

```
cursor.execute('''INSERT INTO Film (Title, Genre, Year)  VALUES ('Wild','Drama',2014); ''')
cursor.execute('''INSERT INTO Film (Title, Genre, Year)  VALUES ('The Lion King','Family',1994); ''')
db.commit()
```

Method 3 - qmark parameters

- Use "?" in the SQL statement to create a parameter query to represent data we do not yet know.
- Replace all the values with question marks (?) and add an additional parameter that will contain the values to be added.

Example of a tuple holding the film data:

```
film  = ('Django Unchained', 'Western', 2012)
cursor.execute("INSERT INTO Film (Title, Genre, Year) VALUES (?, ?, ?);", film)
db.commit()
```

- SQLite expects the values to be in tuple-format. The variable can contain a list if the list items are tuples. E.g.,

```
f = [('Selma', 'Drama', 2014), ('Boyhood', 'Family', 2014)]
```

- In this case, instead of the execute function, we'll want to use the executemany function:

```
cursor.executemany("INSERT INTO Film (Title, Genre, Year) VALUES (?, ?, ?);", f)
db.commit()
```

Searching for data

- To select all data from the film table. The * symbol is known as a wildcard.

```
SELECT * FROM Film
```

- To select certain films include WHERE. Here we want films made in 2014:

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

- To only show certain fields 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, put the SQL statement inside a cursor.execute() statement. You can now use these 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.

The Python code for selecting all records is:

```
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:

```
sql= "Select * from Film where FilmID = 1"
cursor.execute(sql)
print(cursor.fetchone())
```

The next program gives an example of asking the user for a particular genre.

```
choose_genre = input("What genre would you like to see?  ").title()


#question mark (?) in the query is the placeholder
sql= "Select * from Film where genre = ?"
cursor.execute(sql, (choose_genre,))
print(cursor.fetchall())
```

Deleting data

To delete data in the table we could use:

```
cursor.execute("DELETE FROM Film WHERE Year='1994';")
db.commit()
```

Updating data

To update data in the table we could use:

```
cursor.execute("UPDATE Film SET Year='2013' WHERE Title = 'Wild'")
db.commit()
```

Closing the database

To close the database connection when you have finished: `db.close()`