

# **Introduction to Sqlite3 Python Module : Part 1**

**CMPUT 291**

**Introduction to File and Database Management Systems  
University of Alberta  
Department of Computing Science**

# What is sqlite3?

- A Python module for SQLite databases.
- You can connect to a SQLite database and run SQL commands as you did in the shell!
- First, the module should be imported into the python code:
  - `import sqlite3`
- To use the module, create a **connection** object that represents the database
  - `conn = sqlite3.connect('./movie.db') # Creates or opens a  
# database in the path!`
  - `conn = sqlite3.connect(':memory:') # Creates a database in RAM!`

# How to execute a query?

- We need to create a **curser** object and use its **execute()** method
  - `c = conn.cursor()`
  - `c.execute('''CREATE TABLE movie (title TEXT, movie_number INTEGER, PRIMARY KEY (title));'''')`
- **conn.commit()**
  - To Save (**commit**) the changes
  - This method commits the current transaction. If you don't call this method, anything you did since the last call to **commit()** is not visible from other database connections.
  - If you wonder why you don't see the data you've written to the database, double check you call this method.

# Some queries

```
c.execute('''ALTER TABLE movie ADD year INTEGER;''')  
c.execute('''ALTER TABLE movie ADD duration INTEGER;''')
```

```
conn.commit()
```

```
c.execute('''INSERT INTO movie VALUES  
          ('Spiderman', 1, 2000, 100),  
          ('The Dark Knight', 2, 2008, 152),  
          ('Zootopia', 3, 2016, 108);  
          ''')
```

```
conn.commit()
```

# Use `executescript()` for multiple SQL queries

- `execute()` will raise warnings if it is used for multiple queries at the same time.

```
c.executescript('''ALTER TABLE movie RENAME TO temp;
```

```
CREATE TABLE movie (title TEXT,  
movie_number INTEGER,  
runtime INTEGER, year INTEGER,  
PRIMARY KEY (movie_number));
```

```
INSERT INTO movie  
SELECT title, movie_number, duration, year FROM temp;  
DROP TABLE temp;'''")
```

```
conn.commit()
```

# How to select results stored in a table?

- There are two ways to have condition variables within a select query:
  1. Use '?' placeholder and define conditions as a list of tuples.

```
movie_number=(1,)  
c.execute('SELECT * FROM movie WHERE movie_number=?', movie_number)
```

2. Use the named placeholders

```
movie_number=1  
movie_year=2000  
c.execute("SELECT * FROM movie WHERE movie_number=:num and  
year=:year", {"num":movie_number, "year": movie_year})
```

# How to retrieve results from the select?

- **fetchone()** returns just one row as a tuple.

```
row=c.fetchone()  
print row[0]      # prints the title of the row
```

- **fetchall()** returns all rows of the result as a list of tuples

```
c.execute("SELECT * FROM movie;")  
rows=c.fetchall()  
print(rows)      # prints all rows with columns as a list of tuples!  
---  
[(u'Spiderman', 1, 100, 2000), (u'The Dark Knight', 2, 152, 2008),  
(u'Zootopia', 3, 108, 2016)]
```

# Convenient way to access columns

- Efficient way to retrieve columns of each row using their names:
- Set `row_factory` of the connection object (`conn`) to `sqlite3.Row`

```
conn.row_factory = sqlite3.Row
c = conn.cursor()
c.execute("SELECT * FROM movie;")
row = c.fetchone()
print(row.keys())      # ['title', 'movie_number', 'runtime', 'year']
rows = c.fetchall()
for each in rows:
    print (each["movie_number"], each["title"], each["year"])
```

# Final Part

- Don't forget to call the final `commit()` and `close()` the connection.

```
conn.commit()  
conn.close()
```

- OS will commit and close the connection for you, but that is not a good way of coding!
- For the foreign key constraint in SQLite, run its command right after you create a connection

```
import sqlite3  
conn = sqlite3.connect('./movie.db')  
c = conn.cursor()  
c.execute('PRAGMA foreign_keys=ON;')
```

# Example

You are building an employee task management system.

## 1. Setup Database and Insert Data

- Create 'employees' and 'tasks' tables.
- Insert predefined employees and tasks.

## 2. Fetch Data

- Implement functions to Retrieve records.

## 3. Modify Data

- Update and Delete records.

## 4. Final Deliverable

- Generate a summary report of employees and their tasks.
- Implement a function to count tasks per employee.

# Example

This example focuses on using *SQLite3* in Python to-

- Execute queries using '**execute()**'
- Use '**executescript()**' for multiple statements
- Select records using '?' placeholders and/or named placeholders
- Fetch data using '**fetchone()**' and '**fetchall()**'
- Access column names using '**.keys()**' . . .

# Example

Download **sqlite3-example-1.py** from *canvas*!

- You are provided with a Python file that contains several function definitions.
- Some functions are not implemented with a **TODO comment**

**Your task:**

- Fill in the correct logic for each of these functions
- Ensure all queries return expected results
- Do not change the names, arguments, or return types of the provided functions

# Resources

1. <https://docs.python.org/2/library/sqlite3.html>
2. <https://www.sqlite.org/docs.html>