Module 08 – Database Connectivity

tafeSA

Government of South Australia

What is SQLite

- **What is SQLite:-**
- SQLite is an Open Source Database.
-
- SQLite supports standard relational database features like SQL syntax, transactions and prepared statements.

- In addition it requires only little memory at runtime (approx. 250 KByte).

SQLite to Python Data Mapping

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

sqlite3 Module

- To use the sqlite3 module in your Python applications:

  import sqlite

- To use the module you must first create a **connection** object that represents the database:

```
import sqlite3
conn = sqlite3.connect('example.db')
```

- To create a database in RAM use the '**:memory:**' instead of the database name
- Once you have a connection you can create a **cursor** object and call its **execute()** to perform SQL commands

Common Database Operations

Operation	Sqlite3 command
Create a connection to a database	`conn = sqlite3.connect(filename)`
Create a cursor for a connection	`Cursor = conn.cursor()`
Execute a query with the cursor	`cursor.execute(query)`

Return the results of a query	`cursor.fetchall(),cursor.fetchmany(num_rows), cursor.fetchone()`
	`for row in cursor:`
	`    ....`

Commit a transaction to a database	`conn.commit()`
Close a connection	`conn.close()`

Creating a Database

```
import sqlite3

# Create a database in RAM
db = sqlite3.connect(':memory:')

# Creates or opens a database file called
users_db with a SQLite3 DB
db = sqlite3.connect('users_db')
```

Inserting Records (using Placeholder (?)

```
import sqlite3

# Create a database in RAM
db = sqlite3.connect(':memory:')

# Creates or opens a database file called users_db with a SQLite3 DB
db = sqlite3.connect('users_db')
cursor = db.cursor()
name = 'Dale'
phone = '123456'
email = 'dale@example.com'
password = '999_999'
cursor.execute('''INSERT INTO users(name, phone, email, password)
                  VALUES(?,?,?,?)''', (name,phone, email, password))
db.commit()
```

If you need values from Python variables it is recommended to use the "?" placeholder

Never use string operations or concatenation to make your queries because is very insecure.

tafeSA Government of South Australia

Inserting Records (using dictionary)

```
cursor = db.cursor()
name = 'John'
phone = '98987'
email = 'john@example.com'
password = '111_999'

cursor.execute('''INSERT INTO users(name, phone, email, password)
                  VALUES(:name,:phone, :email, :password)''',
                  {'name':name, 'phone':phone, 'email':email, 'password':password})
db.commit()
```

Inserting Several Records -executemany()

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

```
users = [('Sam','11111', 'sam@b.com', 'password3'),
        ('Jim','22222', 'jim@b.com', 'password4'),
        ('Harry','33333', 'harry@b.com', 'password5'),
        ('Mary','44444', 'mary@b.com', 'password6')]

cursor.executemany(''' INSERT INTO users(name, phone, email,
                                         password) VALUES(?,?,?,?)''', users)
db.commit()
```

```
# If you need to get the id of the row you just inserted use lastrowid:
id = cursor.lastrowid
print(f'Last row id: {id}')
```

tafeSA Government of South Australia

Selecting Records

```
To retrieve data, execute the query against the cursor object
and then use fetchone() to retrieve a single row or
fetchall() to retrieve all the rows.
'''
cursor.execute('SELECT name, email, phone FROM users')
user1 = cursor.fetchone() #retrieve the first row
print(user1[0])
all_rows = cursor.fetchall()
for row in all_rows:
    ''' row[0] returns the first column in the query (name),
        row[1] returns email column.
    '''
    print('{0} : {1}, {2}'.format(row[0], row[1], row[2]))
# The cursor object works as an iterator, invoking fetchall() automatically:
cursor.execute('''SELECT name, email, phone FROM users''')
for row in cursor:
    print('{0} : {1}, {2}'.format(row[0], row[1], row[2]))
```

Filtering Records

```
# To retrieve data with conditions, use again the "?" placeholder:
user_id = 3
cursor.execute('''SELECT name, email, phone FROM users WHERE id=?''', (user_id,))
user = cursor.fetchone()
db.commit()
```

Updating|Deleteing Records

```
# The procedure to update data is the same as inserting data:
newphone = '3113093164'
userid = 1
cursor.execute('''UPDATE users SET phone = ? WHERE id = ? ''',
               (newphone, userid))
db.commit()
# The procedure to delete data is the same as inserting data:
delete_userid = 2
cursor.execute('''DELETE FROM users WHERE id = ? ''',
               (delete_userid,))
db.commit()
```

About commit() and rollback()

```
# About commit() and rollback():
'''
Using SQLite Transactions:
Transactions are an useful property of database systems.
It ensures the atomicity of the Database.
Use commit to save the changes.
Use rollback to roll back any change to the database since
the last call to commit:
'''
cursor.execute('''UPDATE users SET phone = ? WHERE id = ? ''',
               (newphone, userid))
# The user's phone is not updated
db.rollback()

'''
Please remember to always call commit to save the changes.
If you close the connection using close or the connection to
the file is lost
(maybe the program finishes unexpectedly), not committed changes
will be lost.
'''
```

Drop a Table

```
db = sqlite3.connect('users_db')
cursor = db.cursor()
cursor.execute('''DROP TABLE users''')
db.commit()

# When we are done working with the DB we need to close the connection:
db.close()
```

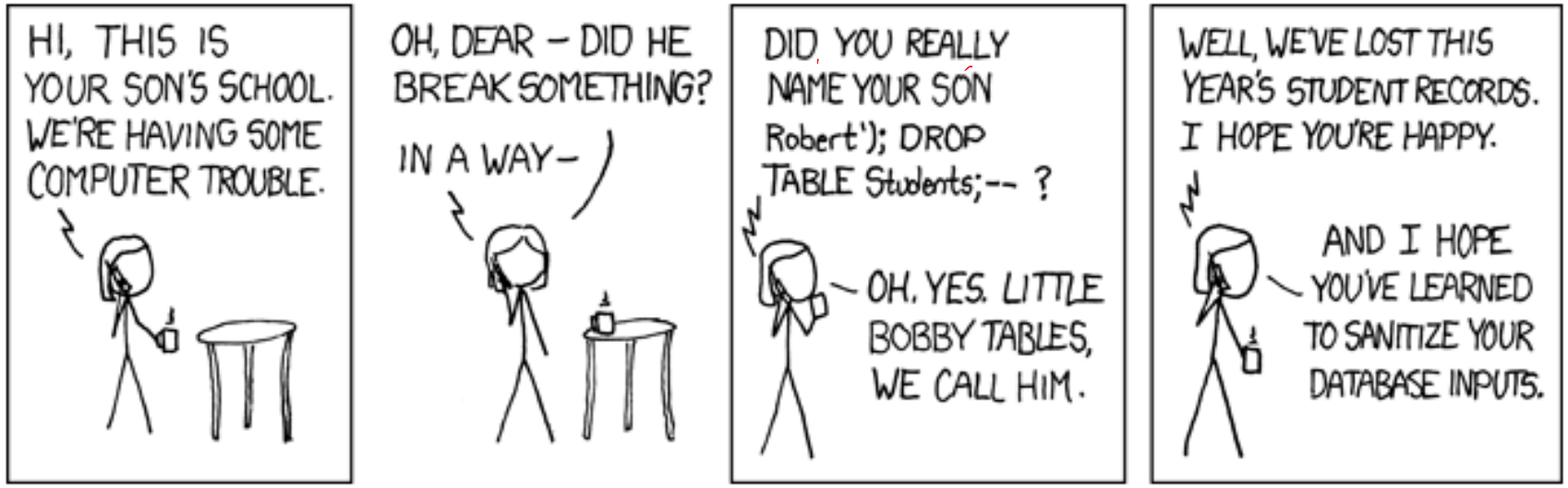
Handling Exceptions

```
try:
    db = sqlite3.connect('users3_db')
    cursor = db.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS
                      users(id INTEGER PRIMARY KEY, name TEXT,
                      phone TEXT, email TEXT, password TEXT)''')
    db.commit()
except Exception as e:
    # This is called a catch-all clause.
    # This is used here only as an example.
    # In a real application you should catch a specific exception
    # such as IntegrityError or DatabaseError

    # Roll back any change if something goes wrong
    db.rollback()
    raise e
finally:
    db.close()
```

Best Practices

```
# Never do this -- insecure
empl_id = '1234'
cur.execute("SELECT * FROM TEmployees WHERE id = '%s'" % empl_id)
print(cur.fetchone())
# Do this instead
empl_id = ('1234',)
cur.execute('SELECT * FROM TEmployees WHERE id=?', empl_id)
print(cur.fetchone())

conn.close()
```

Python SQL Injection



Python SQL Injection (Cont'd)

```
# Never do this -- insecure
empl_id = '1234'
cur.execute("SELECT * FROM TEmployees WHERE id = '" + empl_id + "'")
print(cur.fetchone())
cur.execute("SELECT * FROM TEmployees WHERE id = '%s'" % empl_id)
print(cur.fetchone())
cur.execute("SELECT * FROM TEmployees WHERE id = '{}'".format(empl_id))
print(cur.fetchone())
cur.execute(f"SELECT * FROM TEmployees WHERE id = '{empl_id}'")
print(cur.fetchone())

# Do this instead
cur.execute("SELECT * FROM TEmployees WHERE id=?", (empl_id,))
print(cur.fetchone())
cur.execute("SELECT * FROM TEmployees WHERE id=:Id", {'Id':empl_id})
print(cur.fetchone())
conn.close()
```

Lab Activities

Open Module 08 Lab Exercises Document and complete all lab activities