More on Sqlite3 Python Module

CMPUT 291

Introduction to File and Database Management Systems

University of Alberta

Department of Computing Science

Single query with different values

- executemany() runs a single query with different values for its parameters.
- Inserting a list of values into the database:
- insertions = [('Spiderman2', 4, 2002, 200),
('The Dark Knight2', 5, 2010, 160), ('Zootopia2', 6, 2018, 208),]
- cur.executemany(" INSERT INTO movie VALUES (?,?,?,?) ", insertions);

- Wait! Inserted values are fake!
- conn.rollback()
- rollback() rolls back any changes to the database since the last call to commit()

Be careful about SQL injection!

- Username = get_username_from_user()
- Password = get_password_from_user()
- You want to check whether the user is valid or not. So, you check the database.
  - If you concatenate variables using python string operators (%s, %):
  - cur.execute("SELECT * FROM users WHERE USERNAME='%s' and Password='%s' ;" % username, password)
  - Then you can check the validity of a normal user:
- SELECT * FROM users WHERE USERNAME= 'root' and Password='root123';

The above query is executed by the database. If you get just one row as a result, it means the user is valid and the end user is the 'root'.

How about an attacker?

- Username = root
- Password = attacker' or '1'='1
- The final SQL query that will be executed by the database:
  - SELECT * FROM users WHERE USERNAME='root' and Password='attacker' or '1'='1';
- The attacker logins as the root of the system!
- There are many types of SQL injection attacks and even tools that automatically attack to a database from a single login webpage.
- sqlmap is a tool that can even show the column names of the tables!

Simple ways to prevent SQL injections

- Simply never use python string operators (%s,%) to pass variables into a SQL query.
- Use named or ? placeholders in execute()
- Use regular expressions for checking variables before passing them into a query.
  - For example, a username and password can only contain alphabets, numbers and underscore. Check them not to have any other characters!
- import re
- …
- username=get_username_from_user()
- password=get_password_from_user()

- if re.match("^[A-Za-z0-9_]*$", username) and re.match("^[A-Za-z0-9_]*$", password):
  c.execute('SELECT * FROM users WHERE username=? and password=?;' , (username, password))

Finding Errors

- **complete_statement()** checks that the input is a complete SQL query ending with semi colons.
  - Example:

```
if sqlite3.complete_statement(text):
    cur.execute(text)
```

- Try-catching for errors:

```
try:
        cur.execute(text)
except sqlite3.Error as e:
print 'Error:', e.args[0]
```

A terminal with python for SQLite!

Read the code and explain it, try to implement and run some SQL queries.

```
conn = sqlite3.connect(":memory:")
cur = conn.cursor()
buffer = ""
print "Enter your SQL commands to execute in sqlite3."
print "Enter a blank line to exit."
while True:
	line = raw_input()
	if line == "":
		break
	buffer += line
	if sqlite3.complete_statement(buffer):
		try:
			buffer = buffer.strip()
			cur.execute(buffer)
			conn.commit()

			if buffer.lstrip().upper().startswith("SELECT"):
				print cur.fetchall()
		except sqlite3.Error as e:
			print "An error occurred:", e.args[0]

		buffer = ""
conn.close()
```

User-Defined SQL functions

conn.create_function(name, num_params, func)

Creates a user-defined function that can be used inside SQL queries.

name: The name of the function which will be used in SQL queries.

num_paramas: The number of the parameters it will take in SQL queries.

func: The function in python that actually implements the name.

Example:

1. **Define a function that encrypts the password before inserting into DB.**
2. **We can use the same function for password checking.**

Encrypt Passwords

```
import hashlib
def encrypt(password):
    alg = hashlib.sha256()
    alg.update(password.encode("utf-8"))
    return alg.hexdigest()

conn = sqlite3.connect(":memory:")
conn.create_function("hash", 1, encrypt)
cur = conn.cursor()
data = (username, password, name, address)
cur.execute(" INSERT INTO member (username, password, name, address)
    VALUES (?, hash(?), ?, ?) ", data );
```

Check Passwords

```
import hashlib
def encrypt(password):
    alg = hashlib.sha256()
    alg.update(password.encode("utf-8"))
    return alg.hexdigest()

conn = sqlite3.connect(":memory:")
conn.create_function("hash", 1, encrypt)
cur = conn.cursor()

data = (password, )
cur.execute(" SELECT address FROM member WHERE password LIKE hash(?) ", data);
```

We get the address if the hash of the entered password is what we have in DB.

Some Meta Data

In cursor:

description: the name of the columns

It returns a 7-tuple for each column where the last six items of each tuple are None. (for compatibility reasons!)

```
cur.execute(" SELECT * from member; ")
print "name of the first column: " + cur.description[0][0]
```

In connection:

conn.row_factory

This can be set to a function by which we can define more advanced ways of returning results.

Row Factory

Let's return results as a dictionary of column names:

```
def dictionary_factory (cursor, row):                #Always takes these two arguments.
        dict = {}
        for i, col in enumerate(cursor.description):
                dict[col[0]] = row[i]
        return dict

conn= sqlite3.connect(":memory:")
conn.row_factory = dictionary_factory                #Set it before creating a cursor object
cur = conn.cursor()

cur.execute(" SELECT * from member; ")
result = cur.fetchone()
print " the first column: "
print result['username']
```

By Setting conn.row_factory = sqlite3.Row, we can also retrieve the results with the column names.

Null Handling

Querying:

The operators *IS NULL* and *IS NOT NULL* may be used in queries inside SQLITE

Updating:

```
cur.execute("INSERT INTO table_name
VALUES('att_value1',null);")
```

Null Handling

To only return full defined record:

```
cur.execute("SELECT att_name FROM table_name WHERE att_name IS NOT NULL;")
```

To return records that contain null(s):

```
cur.execute("SELECT att_name FROM table_name WHERE att_name IS NULL;")
```

Note: This will require some prior knowledge of what attributes may have missing values.

Null Handling

```
INSERT INTO movie VALUES
   ...> ('The Matrix',1,2000,120),
   ...> ('Hello',2,2016,128),
   ...> ('lalaland',3,null,null);
```

Python NONE data type ≈ SQLite NULL

```
c=conn.cursor()
cur.execute('SELECT * FROM movie;')
movies = c.fetchall()
for movie in movies:
    for attribute in movie:
        if attribute == None:
            print(movie)
            Break
```

OUTPUT:

```
('La La Land', 3, None, None)
```

Continuing the Example

- Schema:
  - course (course_id, title, seats_available)
  - student (student_id, name)
  - enroll (student_id, course_id, enroll_date, grade)
- Our department offers some courses and we have a table for the students.
- Every student can register in a course.
- Students can drop courses.
- The system keeps track of the grades for each student in every course.

Example

1. Download sqlite3-example2.py from e-class!
2. Read the code!

3. Complete the drop function which drops a course for a student.

4. Define the SQL GPA function which maps each grade to a numerical value
    1. Grade='A' -----> GPA=4
    2. Grade='B' -----> GPA=3
    3. Grade='C' -----> GPA=2
    4. else -----> GPA=0
5. Use the GPA function to get a sorted list of the student names with their average GPAs.

Example

```
def drop(student_id, course_id):
    global connection, cursor

    # Drop the course for the student and update the seats_avialable column

    connection.commit()
    return

def GPA(grade):
    # Map the grade to a numerical value
    return 0
```

Example

```
def main():
    global connection, cursor
path="./register.db"
connect(path)
connection.create_function('GPA', 1, GPA)
define_tables()
insert_data()
enroll_assign_grades()

# Use the GPA function to get a sorted list of the student names with their average GPAs.

connection.commit()
connection.close()
    return
```

Resources

1. https://docs.python.org/2/library/sqlite3.html
2. https://www.sqlite.org/docs.html
3. http://sqlmap.org/
4. https://docs.python.org/2.7/library/hashlib.html

Example

```
def drop(student_id, course_id):
    global connection, cursor

    data = (student_id, course_id)
    cursor.execute('DELETE FROM enroll WHERE student_id=? and course_id=?;', data)

    data = (course_id, student_id, course_id)
    cursor.execute( ''' UPDATE course SET seats_available = seats_available + 1
     where course_id=? and NOT EXISTS
                (select * from enroll WHERE student_id=? and course_id=?); ''' ,data)

    connection.commit()
    return
```

Example

```
def GPA(grade):
    if grade=='A':
        return 4
    elif grade=='B':
        return 3
    elif grade=='C':
        return 2
    return 0
```

Example

```
def main():
        global connection, cursor
path="./register.db"
connect(path)
connection.create_function('GPA', 1, GPA)
define_tables()
insert_data()
enroll_assign_grades()

cursor.execute('''
        SELECT s.name, AVG(GPA(e.grade)) AS avg_gpa
        FROM student AS s, enroll AS e
        WHERE s.student_id = e.student_id
        GROUP BY s.name
        ORDER BY avg_gpa;''')

all_entry = cursor.fetchall()
for one_entry in all_entry:
print one_entry
print

connection.commit()
connection.close()
        return
```