Ch26-SlqiteDB

August 7, 2020

1 Sqlite database

- https://www.sqlite.org/
- C-based, one of the most used embedded database (zero configuration)

1.1 SQL basics

- Structured Query Language
- case insensitive language; usually written in uppercase
- let's you or program use SQL-based databases such as SQLite, MySQL, MSSQL, PostgreSQL, etc.
- most important basic statents to learn: CRUD
- C: create (database, table, create and insert records)
- R: retrieve/read data
- U: update data
- D: delete data
- http://www.w3schools.com/sql/default.asp

1.2 sqlite browser

- GUI-based sqlite db explorer
- makes it easy to see data and learn SQL
- http://sqlitebrowser.org/

1.3 sqlite3 module

- python3 provides sqlite3 library to work with sqlite database
- https://docs.python.org/3/library/sqlite3.html
- SQLite natively supports the following types: NULL, INTEGER, REAL, TEXT, BLOB

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

1.4 in memory db example

[1]:
```
import sqlite3
# connect to the memory database
con = sqlite3.connect(":memory:")

# create a table
con.execute("create table person(fname, lname)")
```

[1]:
```
<sqlite3.Cursor at 0x109d803b0>
```

[2]:
```
# fill the table with data
persons = [('Hugo', 'Boss'), ('Calvin', 'Klien')]
con.executemany("insert into person(fname, lname) values (?, ?)",
                persons)
```

[2]:
```
<sqlite3.Cursor at 0x109d80570>
```

[3]:
```
# print the table contents
for row in con.execute("select rowid, fname, lname from person"):
    print(row)
```

```
(1, 'Hugo', 'Boss')
(2, 'Calvin', 'Klien')
```

[4]:
```
print("I just deleted", con.execute("delete from person where rowid=1").
 ↪rowcount,
      "rows")
```

```
I just deleted 1 rows
```

1.5 db file example

1.5.1 create database, create table and insert data into table

[5]:
```
import sqlite3
# create connection
conn = sqlite3.connect('example.db')
# create cursor object
cur = conn.cursor()

cur.execute("""CREATE TABLE IF NOT EXISTS students (
                firstName text,
                lastName text,
                test1 real,
                test2 real,
                average real,
                grade text
```

```
            )
        """)
```

[5]: <sqlite3.Cursor at 0x109d808f0>

[6]:
```
query = """ INSERT INTO students (firstName, lastName,
            test1, test2) values (?, ?, ?, ?)
        """
cur.execute(query, ('John', 'Smith', 99, 95.5))
```

[6]: <sqlite3.Cursor at 0x109d808f0>

[7]:
```
cur.execute(query, ('Michael', 'Jordan', 50, 65))
```

[7]: <sqlite3.Cursor at 0x109d808f0>

[8]:
```
# save/commit the changes to the db
conn.commit()
# close the database if done
conn.close()
```

1.5.2 open database, read and update table

[9]:
```
import sqlite3
conn = sqlite3.connect('example.db')
cur = conn.cursor()
```

[10]:
```
cur.execute('SELECT * FROM students where rowid = 1')
row = cur.fetchone() # returns one row as tuple if rowid with value 1 exists
print(row)
```

```
('John', 'Smith', 99.0, 95.5, None, None)
```

[11]:
```
for col in row:
    print(col)
```

```
John
Smith
99.0
95.5
None
None
```

[12]:
```
cur.execute('SELECT rowid, * FROM students')
rows = cur.fetchall()
print(type(rows))
```

```
<class 'list'>
```

[13]:
```
for row in rows:
    print(row)
```

```
(1, 'John', 'Smith', 99.0, 95.5, None, None)
(2, 'Michael', 'Jordan', 50.0, 65.0, None, None)
```

update table

[14]:
```
for row in rows:
    avg = (row[3] + row[4])/2
    # grade = ?
    cur.execute('update students set average=? where rowid=?', (avg, row[0]))
```

[15]:
```
cur.execute('select * from students')
print(cur.fetchall())
```

```
[('John', 'Smith', 99.0, 95.5, 97.25, None), ('Michael', 'Jordan', 50.0, 65.0,
57.5, None)]
```

[16]:
```
# commit changes and close connection
conn.commit()
conn.close()
```

1.6 SQL Injection Vulnerability

- how not to write sql query in programs

[17]:
```
import sqlite3
conn = sqlite3.connect('example.db')
cur = conn.cursor()

cur.execute("""CREATE TABLE IF NOT EXISTS users (
                username text unique,
                password text
                )
            """)
```

[17]:
```
<sqlite3.Cursor at 0x109d80730>
```

[18]:
```
# Prompt user to create account
username = input('Enter your username: ')
password = input('Pick a password: ')
```

```
Enter your username: john
Pick a password: password
```

[23]:
```
# bad passwords
# insecure way to create sql statements
sqlinsert = "insert into users (username, password) values ('{0}', '{1}')".
 ↪format(username, password)
print(sqlinsert)
cur.execute(sqlinsert)
```

```
insert into users (username, password) values ('john', 'password')
```

[23]:
```
<sqlite3.Cursor at 0x109d80730>
```

[24]:
```
# check database
conn.commit()
for row in cur.execute('select * from users'):
    print(row)
```

```
('john', 'password')
```

1.6.1 what is wrong with the above codes?

1.6.2 authenticate users and SQL injection attack

[25]:
```
# Prompt user to create account
def insecureAuthentication():
    username = input('Enter your username: ')
    password = input('Pick a password: ')
    sqlSelect = "select * from users where username = '{0}' \
                    and password = '{1}'".format(username, password)
    cur.execute(sqlSelect)
    row = cur.fetchone()
    if row:
        print('Welcome {}, this is your kingdom!'.format(row[0]))
    else:
        print('Wrong credentials. Try Again!')
```

[26]:
```
insecureAuthentication()
```

```
Enter your username: john
Pick a password: password
Welcome john, this is your kingdom!
```

[30]:
```
# sql injection; authenticate without using password
insecureAuthentication()
```

```
Enter your username: john' or '1'='1
Pick a password: adfadsfdsf
Welcome john, this is your kingdom!
```

1.7 secure way to store password

- https://docs.python.org/3/library/hashlib.html

[31]:
```
import uuid
import hashlib, binascii

def createSecurePassword(password, salt=None, round=100000):
    if not salt:
        salt = uuid.uuid4().hex

    """
    for i in range(round):
        password = password+salt
        password = hashlib.sha256(password.encode('utf-8')).hexdigest()
    """
    # hashlib.pbkdf2_hmac(hash_name, password, salt, iterations, dklen=None)
    dk = hashlib.pbkdf2_hmac('sha256', password.encode('utf-8'),
                         salt.encode('utf-8'), round)
    password = binascii.hexlify(dk)
    return "%s:%s"%(password, salt)
```

[32]:
```
def secureRegistration():
    # Prompt user to create account
    username = input('Enter your username: ')
    password = input('Enter your password: ')
    secPass = createSecurePassword(password)
    insert = 'insert into users (username, password) values (?, ?)'
    cur.execute(insert, (username, secPass))
```

[34]:
```
# register a user
secureRegistration()
```

```
Enter your username: jake
Enter your password: password1
```

[35]:
```
# check data
for row in cur.execute('select * from users'):
    print(row)
```

```
('john', 'password')
('jake', "b'c318988672d05094deaffce0148a49b1b43dfc89f3b8b75d251de60446dcecc5':53
40a4af29574554997b0fe7a1ac670b")
```

[ ]:
```
conn.commit()
```

[36]:
```
def secureAuthentication():
    username = input('Enter your username: ')
    password = input('Enter your password: ')
    # use parameterized query
    sqlSelect = 'select password from users where username = ?'
    cur.execute(sqlSelect, (username,))
    row = cur.fetchone()
    if row:
        # username exists
        # check password hashes
        hashpass = row[0]
        hashedPass = hashpass[:hashpass.find(':')]
        salt = hashpass[hashpass.find(':')+1:]
        secPass = createSecurePassword(password, salt)
        if hashpass == secPass:
            print('Welcome to your kingdom, {}'.format(username))
        else:
            print('Wrong credentials. Try Again!')
    else:
        print('Wrong credentials. Try Again!')
```

[37]:
```
secureAuthentication()
```

```
Enter your username: jake
Enter your password: password1
Welcome to your kingdom, jake
```

[39]:
```
# try the same SQL injection
secureAuthentication()
```

```
Enter your username: jake' or '1' = '1
Enter your password: adsfadsf
Wrong credentials. Try Again!
```

[ ]:
```
conn.commit()
conn.close()
```

[ ]:
```

```