IASP505
Scripting Language for Database Management
[SQLite3 in Python]

Dr. John Yoon

Mercy College

CYBERSECURITY

SQLite3

http://sqlitebrowser.org/

- Download and install it
- Create a database
  - Tables
    - Attributes and types

```
CREATE TABLE artist(
  artistid    INTEGER PRIMARY KEY,
  artistname  TEXT
);
```

  - Relate tables
    - Common attributes → reference to

```
CREATE TABLE track(
  trackid     INTEGER,
  trackname   TEXT,
  trackartist INTEGER,
  FOREIGN KEY(trackartist) REFERENCES artist(artistid)
);
```

SQLite3

- self-contained, serverless, zero-configuration and transactional
- very fast and lightweight
- the entire database stored in a single disk file

Check in Python

- In command-line

```
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 10:22:32) on Windows (64 bits).
This is the Pyzo interpreter with integrated event loop for PYQT5.
Type 'help' for help, type '?' for a list of *magic* commands.

>>> import sqlite3 as sq

>>> sq.version
'2.6.0'

>>> sq.sqlite_version
'3.21.0'
```

Check in SQLite

- In command-line

```
$ sqlite3 test.db
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";“

sqlite> .help

sqlite> .tables
sqlite> .exit

$ dir test.db
```

More SQLite Command

```
sqlite> create table tbl1(one varchar(10), two smallint);
sqlite> insert into tbl1 values('hello!',10);
sqlite> insert into tbl1 values('goodbye', 20);
sqlite> select * from tbl1;
hello!|10 goodbye|20
sqlite>
```

- In command-line

```
$ sqlite3 test.db
SQLite version 3.7.13 2012-06-11 02:05:22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"

sqlite> .mode list
sqlite> select * from tbl1;
hello|10
goodbye|20
sqlite>
```

```
sqlite> .mode column
sqlite> select * from
One          two
---------- ---------
Hello        10
Goodbye      20
sqlite> .width 12 6
```

```
sqlite> .separator ", "
sqlite> select * from tbl1;
Hello, 10
Goodbye, 20
sqlite>
```

More SQLite Command

- Writing results to a file

```
sqlite> .mode list
sqlite> .separator |
sqlite> .output test_file_1.txt
sqlite> select * from tbl1;
sqlite> .exit

$ cat test_file_1.txt
hello|10
goodbye|20 $
```

Using SQL in Python

- There is a tutorial on SQlite in:

  https://docs.python.org/3.6/library/sqlite3.html

Using sqlite3 in Python

- Python includes a simple database implementation called sqlite3 that provides a database with fast disk access.
- These presentation is based on the tutorial provided in:
  https://docs.python.org/3.6/library/sqlite3.html

- To use the database you need to create a connection object first:

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

- If you want the database to be created in memory use ":memory:" instead but your data will not be saved.

How to create a database under a new folder?

Using Python's SQLite Module

- use the SQLite3 module we need to add an import statement to our python script

```
import sqlite3
```

- use the function sqlite3.connect to connect to the database

```
# Create a database in RAM
db = sqlite3.connect(':memory:')
# Creates or opens a file called mydb
db = sqlite3.connect('data/mydb')
```

- When done working with the DB, close the connection

```
db.close()
```

To locate a directory:

```
import os
os.getcwd()
os.chdir()
os.listdir()
os.mkpir()
```

Creating and Deleting Tables

- get a cursor object and pass the SQL statements to the cursor object to execute
- commit the changes finally.

```
# Get a cursor object
cursor = db.cursor()
cursor.execute('''
    CREATE TABLE users(id INTEGER PRIMARY KEY,
           name TEXT, phone TEXT, email TEXT unique,
           password TEXT)
''')
db.commit()
```

- To drop a table

```
# Get a cursor object
cursor = db.cursor()
cursor.execute('''DROP TABLE users''')
db.commit()
```

Inserting (Populating) Data

- To insert data we use the cursor to execute the query
- use the "?" placeholder
- Never use string operations or concatenation to make your queries because is very insecure

```
cursor = db.cursor()
name1 = 'Andres'
phone1 = '3366858'
email1 = 'user@example.com'
# A very secure password
password1 = '12345'

name2 = 'John'
phone2 = '5557241'
email2 = 'johndoe@example.com'
password2 = 'abcdef'
```

```
cursor = db.cursor()
name1 = 'Andres'
phone1 = '3366858'
email1 = 'user@example.com'
# A very secure password
password1 = '12345'

name2 = 'John'
phone2 = '5557241'
email2 = 'johndoe@example.com'
password2 = 'abcdef'

# Insert user 1
cursor.execute('''INSERT INTO users(name, phone,
      email, password) VALUES(?,?,?,?)''',
      (name1,phone1, email1, password1))
print('First user inserted')

# Insert user 2
cursor.execute('''INSERT INTO users(name, phone,
      email, password) VALUES(?,?,?,?)''',
      (name2,phone2, email2, password2))
print('Second user inserted')
db.commit()
```

```
ins = 'INSERT INTO users(name, phone,
      email, password) VALUES(?,?,?,?)'
cursor.execute(ins, (name1,phone1, email1, password1))
```

Inserting (Populating) Data: Alternative

- Another way to do this is passing a dictionary using the ":keyname" placeholder

```
cursor.execute('''INSERT INTO users
        (name, phone, email, password)
         VALUES(:name,:phone, :email, :password)''',
        {'name':name1, 'phone':phone1, 'email':email1,
        'password':password1})
```

Inserting (Populating) bulk Data: Alternative

- If you need to insert several users use `executemany` and a list with the tuples

```
users = [(name1,phone1, email1, password1),
         (name2,phone2, email2, password2),
         (name3,phone3, email3, password3)]

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('Last row id: %d' % id)
```

Using sqlite3 in Python

- It is not recommended to use string functions to assemble your SQL commands since it makes your program vulnerable.
- Instead use sqlite3 parameter substitution.

```
# Never do this -- insecure!
symbol = 'IBM'
c.execute("... where symbol = '%s'" % symbol)

# Do this instead
t = (symbol,)
c.execute('select * from stocks where symbol=?', t)

# Larger example
for t in [('2006-03-28', 'BUY', 'IBM', 1000, 45.00),
          ('2006-04-05', 'BUY', 'MSOFT', 1000, 72.00),
          ('2006-04-06', 'SELL', 'IBM', 500, 53.00),
         ]:
    c.execute('insert into stocks values (?,?,?,?,?)', t)
```

Retrieving Data (SELECT)

- 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])
#Print the first column retrieved(user's name)

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]))
```

Retrieving Data (SELECT)

- as an iterator, invoking `fetchall()` automatically

```
cursor.execute('''SELECT name, email, phone FROM users''')
for row in cursor:
    # 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]))
```

- 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()
```

Updating & Deleting Data

- update or delete data is the same as inserting data

```
# Update user with id 1
newphone = '3113093164'
userid = 1
cursor.execute('''UPDATE users SET phone = ?
      WHERE id = ? ''', (newphone, userid))

# Delete user with id 2
delete_userid = 2
cursor.execute('''DELETE FROM users WHERE id = ? ''',
      (delete_userid,))

db.commit()
```