Database Design

Python Connect SQLite Tutorial

Nguyen Van Dieu, Lecturer

HoChiMinh City University of Transport

dieu.nguyen@ut.edu.vn

2018

Table of contents

String literals and multiline strings

* A string literal, in this way, is a literal which yields a string.
* In Python, those literals are marked by many ways.
* Two ways are to put a single or double quote at either the beginning or the end of the literal:

Single quote

```
'A string literal'
```

Double quote

```
"Another string literal"
```

String literals and multiline strings

* Other ways are to put three single or double quotes in the same positions:

Triple double quote in one line string literals

"""Only one line"""

Triple double quote for many line string literals

"""line 1
...
line n"""

Tripble single quote for many line string literals

'''line 1
...
line n'''

Creating a Database and Making a Connection

* Connect SQLite database; if the database represented by the file does not exists one will be created at that path.

Connect SQLite3

```
import sqlite3
con = sqlite3.connect('/path/to/file/databaseName.db')
```

Creating Tables

Orders(OderID, OrderDate, RequireDate, CustomerID)

OrderDetails(OrderID, ProductID, Price, Quantity)

Make cur and create table: **Order**

```
cur = con.cursor()

ordersSQL = """
        CREATE TABLE Orders  (
           OderID text PRIMARY KEY,
           OrderDate text NOT NULL,
           RequireDate text,
           CustomerID text
        ) """

cur.execute(odersSQL)
```

Creating Tables

Create table: **OrderDetails**

```
orderDetailsSQL = """
        CREATE TABLE OrderDetails  (
            OderID text,
            ProductID text,
            OrderDate text NOT NULL,
            Price real,
            Quantity int,
            PRIMARY KEY (OrderID, ProductID) ) """

cur.execute(orderDetailsSQL)
```

Show all table in database

We can query the sqlite_master table, a built-in SQLite metadata table, to verify that the above commands were successful.

sqlite_master

```
cur.execute(""""SELECT name
                FROM sqlite_master
                WHERE type='table' """")
```

fetchall()

```
print(cur.fetchall())
```

Results:

```
[('Orders',), ('OrderDetails',)]
```

Show Table by SQL

Look at the schema of the tables query the SQL:

sqlite_master

```
cur.execute("""SELECT sql
               FROM sqlite_master
               WHERE type='table'
                  AND name='Orders' """)
```

fechall()

```
print(cur.fetchall()[0])
```

Results:

```
'CREATE TABLE Orders (
    OrderID text primary key, OrderDate text not null,
    RequireDate text, CustomerID text)'
```

Loading the Data

The workflow for executing INSERT statements is simply:

- Connect to the database
- Create a cursor object
- Write a parameterized insert SQL statement and store as a variable
- Call the execute method on the cursor object passing it the sql variable and the values, as a tuple, to be inserted into the table

Loading the Data

Insert Into

```
cur = con.cursor()
odersSQL = """INSERT INTO
Orders( OrderID, OrderDate, RequireDate, CustomerID )
VALUES (?, ?, ?, ?)"""
```

* The insert statement except for the ?. It is known as a "parameterized query".

execute()

```
cur.execute(odersSQL, ('D01', '2018-10-21', '2018-10-27', 'K01'))
cur.execute(odersSQL, ('D02', '2018-10-22', '2018-10-28', 'K02'))
cur.execute(odersSQL, ('D03', '2018-10-23', '2018-10-27', 'K01'))
```

Query Data

execute

```
result = cur.execute("select * from Orders")
for row in result:
    print(row)
```

Result:

```
('D01', '2018-10-21', '2018-10-27', 'K01')
('D02', '2018-10-22', '2018-10-28', 'K02')
('D03', '2018-10-23', '2018-10-27', 'K01')
```

Exception

Python Exception

```
try:
    cur.execute(" Statement ")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
else:
    "doing something if necessary."
```

Example: Exception

Python Exception

```
try:
    cur.execute("insert into Orders
        values('D01', '2018-10-21', '2018-10-27', 'K01')")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
```

Throw error:

Database error: UNIQUE constraint failed: Orders.OrderID

FD is an Integrity Constraints

Suppose $\mathcal{R}(ABC)$ with $A \rightarrow B$ hold on $\mathcal{R}$
There is an *integrity constraint* of scheme R.

Integrity Constraint

- Context: R
- Condition:
  $\forall (t_1, t_2) \in \forall r : t_1.A = t_2.A$
  $t_1.B = t_2.B$
  end.
- Influence table:

	Insert	Delete	Update
R	+	-	+(A/B)

Insert

Create Trigger Insert

```
cur.execute ( """create trigger R_insert after insert on R
begin
   select case
         when ( select count(distinct B)
                from R
                where A = new.A ) > 1 then
           raise (abort, 'Insert violate A → B')
   end;
end; """ )
```

Update A or B

Create Trigger Update

```
cur.execute ( """create trigger R_update after update on R
begin
   select case
         when ( select count(distinct B)
                  from R
                  where A = new.A ) > 1 then
           raise (abort, 'Update violate A → B')
   end;
end; """ )
```

1. Testing Trigger: Insert

Insert Valid

```
try:
    cur.execute("insert into R values (1, 1, 1)")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
else:
    result = cur.execute("select * from R")
    for row in result:
        print(row)
```

Result: Valid $A \rightarrow B$

(1, 1, 1)

2. Testing Trigger: Insert

Insert Valid

```
try:
    cur.execute("insert into R values (1, 1, 2)")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
else:
    result = cur.execute("select * from R")
    for row in result:
        print(row)
```

Result: Valid $A \rightarrow B$

(1, 1, 1)
(1, 1, 2)

3. Testing Trigger: Insert

Insert Invalid

```
try:
    cur.execute("insert into R values (1, 2, 3)")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
else:
    result = cur.execute("select * from R")
    for row in result:
        print(row)
```

Result: *Error from SQLite*

Database error: *Insert violate $A \rightarrow B$*

4. Testing Trigger: Insert

Insert Valid

```
try:
    cur.execute("insert into R values (2, 2, 3)")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
else:
    result = cur.execute("select * from R")
    for row in result:
        print(row)
```

Result: Valid $A \rightarrow B$

(1, 1, 1)
(1, 1, 2)
(2, 2, 3)

5. Testing Trigger: Update A

Update A: Invalid

```
try:
    cur.execute("update R set A = 2 where C = 2")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
else:
    result = cur.execute("select * from R")
    for row in result:
        print(row)
```

Result: *Error from SQLite*

Database error: *Update violate $A \rightarrow B$*

6. Testing Trigger: Update B

Update B: Invalid

```
try:
    cur.execute("update R set B = 2 where C = 1")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
else:
    result = cur.execute("select * from R")
    for row in result:
        print(row)
```

Result: *Error from SQLite*

Database error: *Update violate $A \rightarrow B$*

7. Testing Trigger: Update B

Update A: Valid

```
try:
    cur.execute("update R set A = 3 where C = 3")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
else:
    result = cur.execute("select * from R")
    for row in result:
        print(row)
```

Result: Valid $A \rightarrow B$

(1, 1, 1)
(1, 1, 2)
(3, 2, 3)

8. Testing Trigger: Update B

Update B: Valid

```
try:
    cur.execute("update R set B = 3 where C = 3")
    con.commit()
except sqlite3.Error as e :
    print("Database error: ", e)
else:
    result = cur.execute("select * from R")
    for row in result:
        print(row)
```

Result: Valid $A \rightarrow B$

(1, 1, 1)
(1, 1, 2)
(3, 3, 3)

Exercises


More Complex

Consider $\mathcal{R}_1(ABC), \mathcal{R}_2(CDE)$

$A \rightarrow D$ hold on $\mathcal{R}_1$ and $\mathcal{R}_2$

Check integrity constraint $A \rightarrow D$ on them.