

Week 11 HIA

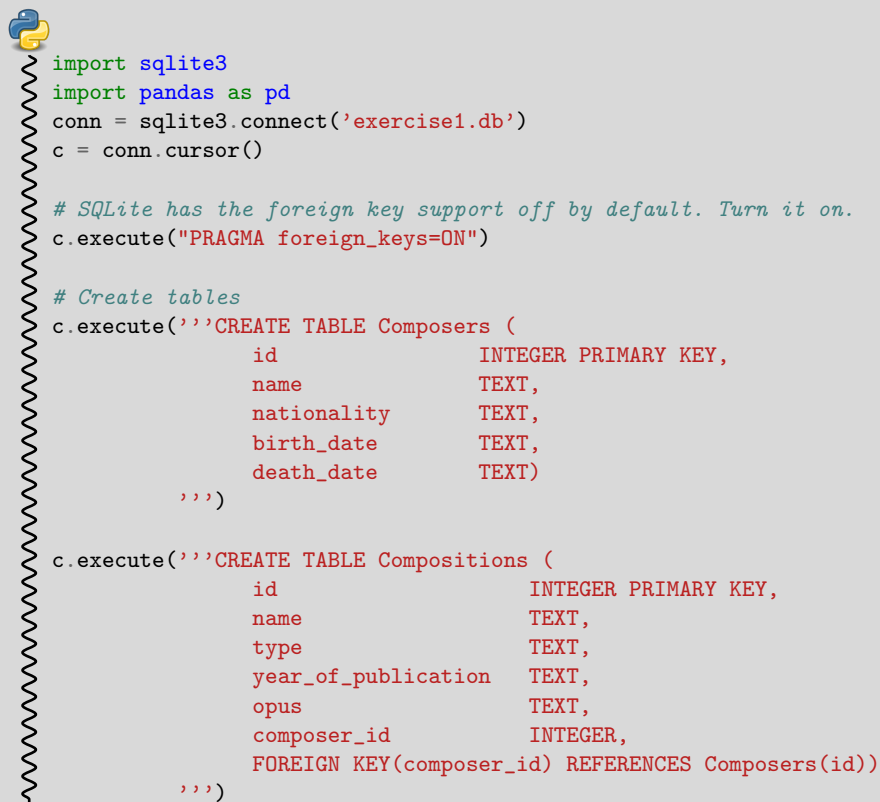
Terry Yin

November 17, 2014

This hand-in assignment is using IPython Notebook [Pérez & Granger \(2007\)](#), with the support from SQLite database [Owens & Allen \(2006\)](#), and the Pandas library [McKinney \(2010\)](#).

1 HIA Part I: Database Design

I use SQLite3 as my database. And the following Python code will create the 2 required tables. After creating the tables. The next Python code will add some data to both tables.

A screenshot of a Jupyter Notebook cell showing Python code to create two SQLite tables. The code imports sqlite3 and pandas, connects to a database named 'exercise1.db', and enables foreign key support. It then creates a 'Composers' table with columns id, name, nationality, birth_date, and death_date. Finally, it creates a 'Compositions' table with columns id, name, type, year_of_publication, opus, composer_id, and a foreign key reference to the Composers table.

```
import sqlite3
import pandas as pd
conn = sqlite3.connect('exercise1.db')
c = conn.cursor()

# SQLite has the foreign key support off by default. Turn it on.
c.execute("PRAGMA foreign_keys=ON")

# Create tables
c.execute('''CREATE TABLE Composers (
            id                INTEGER PRIMARY KEY,
            name              TEXT,
            nationality        TEXT,
            birth_date        TEXT,
            death_date        TEXT)
        ''')

c.execute('''CREATE TABLE Compositions (
            id                INTEGER PRIMARY KEY,
            name              TEXT,
            type              TEXT,
            year_of_publication TEXT,
            opus              TEXT,
            composer_id       INTEGER,
            FOREIGN KEY(composer_id) REFERENCES Composers(id))
        ''')
```

Out[12]: <sqlite3.Cursor at 0x112b0a1f0>



```
# Insert composers
c.execute('INSERT INTO Composers VALUES (1,\'Joseph Haydn\',\'Austria\',
\'31 Mar 1732\',\'31 May 1809\')')
c.execute('INSERT INTO Composers VALUES (2,\'Wolfgang Amadeus Mozart\',\'Austria\',
\'27 Jan 1756\',\'5 Dec 1791\')')
c.execute('INSERT INTO Composers VALUES (3,\'Christoph Graupner\',\'German\',
\'13 Jan 1683\',\'10 May 1760\')')
c.execute('INSERT INTO Composers VALUES (4,\'Franz Peter Schubert\',\'Austria\',
\'31 Jan 1797\',\'19 Nov 1828\')')
c.execute('INSERT INTO Composers VALUES (5,\'Tomaso Giovanni Albinoni\',\'Italy\',
\'8 Jun 1671\',\'17 Jan 1751\')')
c.execute('INSERT INTO Composers VALUES (6,\'Carl Philipp Emanuel Bach\',\'German\',
\'8 Mar 1714\',\'14 Dec 1788\')')

# Insert compositions
c.execute("INSERT INTO Compositions VALUES (1,\'Cello Concerto No. 2 in D\',
\'concertos\',\'1783 \',\'op.101\', 1)")
c.execute("INSERT INTO Compositions VALUES (2,\'Apollo and Hyacinth\',
\'opera\',\'1767 \',\'k.38\', 2)")
c.execute("INSERT INTO Compositions VALUES (3,\'Bastien and Bastienne\',
\'opera\',\'1768 \',\'k.50\', 2)")

# Save (commit) the changes
conn.commit()
```



```
> pd.read_sql("SELECT * from Composers", conn)
```

```
Out[14]:
```

	id	name	nationality	birth_date	death_date
0	1	Joseph Haydn	Austria	31 Mar 1732	31 May 1809
1	2	Wolfgang Amadeus Mozart	Austria	27 Jan 1756	5 Dec 1791
2	3	Christoph Graupner	German	13 Jan 1683	10 May 1760
3	4	Franz Peter Schubert	Austria	31 Jan 1797	19 Nov 1828
4	5	Tomaso Giovanni Albinoni	Italy	8 Jun 1671	17 Jan 1751
5	6	Carl Philipp Emanuel Bach	German	8 Mar 1714	14 Dec 1788



```
pd.read_sql('SELECT Compositions.name,
Compositions.type,
Composers.name as \'composer name\'
FROM Compositions JOIN Composers
WHERE composer_id = Composers.id
', conn)
```

```
Out[4]:
```

	name	type	composer name
0	Cello Concerto No. 2 in D	concertos	Joseph Haydn
1	Apollo and Hyacinth	opera	Wolfgang Amadeus Mozart
2	Bastien and Bastienne	opera	Wolfgang Amadeus Mozart



```
? conn.close()
```

2 HIA Part II: Who Asked These Questions

2.1 How data could be most efficiently stored on a disk

This question is more likely to be asked by the database administrator ([Wikipedia 2014a](#), DBA). DBA is responsible for the installation, configuration, upgrading, administration, monitoring, maintenance, and security of databases in an organization. Each hard disk has limited size. For larger database, the DBA need to decide how to balance the use of the physical hard disks. When the database grew even larger, DBA need to decide how to partition the database, [Wikipedia \(2014b\)](#).

2.2 If there is a vacancy on Flight 243

This question is asked by the end-user.

2.3 How a relation could be stored as a sequential file

This is the concern of the programmer of application software. The choice of using a sequential file ([Brookshear 2011](#), 406), typically, doesn't involve any DBMS. Programmers need to decide by themselves on how to structure the file. It's like creating a Do-It-Yourself database.

2.4 How many times a user should be allowed to mistype a password before a conversation is terminated.

This sounds like a question asked by the programming of application software. But it should be decided by one type of end-user, the operator, or owner of the business. Password re-trying is a feature of the application. So, user should decide that. But as the programmers are software experts, they might have more meaningful suggestions on how to balance security and flexibility. And last, the DBA knows better how strong the password is secured in the database, so they can also give suggestions on this.

2.5 How the PROJECT operation (of the relational model) can be implemented

The designer of database management system software need to ask this question. Projection is a concept from relational algebra [Wikipedia \(2014c\)](#). It means collecting a subset of columns [394]brookshear1997computer">. The DBMS designer need to think about how to implement it in a real life database. For example, in the SQL for MySQL database, both selection and projection are done by the SELECT command.

```
SELECT name, gender, age
FROM students
WHERE grade = 1
```

Will do both selection on the rows and the projection on columns.

The designer of the DBMS need also to decide where to put the result of the PROJECT operation. It's a subset of the original table and change on the PROJECT result should reflect on the original table in real-time.

3 HIA Part III:

Below are the two tables (Employees and Department) implemented in SQLite database.

```

!rm exercise3.db
conn = sqlite3.connect('exercise3.db')
c = conn.cursor()

# Create tables
c.execute('''CREATE TABLE Employees (
            Name            TEXT,
            Email_address   TEXT,
            Location        INTEGER)''')
c.execute('''CREATE TABLE Department (
            Manager_Name    TEXT,
            Floor_Number    INTEGER)''')

# Insert data
c.execute('INSERT INTO Employees VALUES ("John", "John.john@abxz.com", 5)')
c.execute('INSERT INTO Employees VALUES ("Mary", "m.smith@abxz.com", 3)')
c.execute('INSERT INTO Employees VALUES ("Philip", "philipx@abxz.com", 5)')
c.execute('INSERT INTO Department VALUES ("Karen", 4)')
c.execute('INSERT INTO Department VALUES ("David", 5)')

# Save (commit) the changes
conn.commit()

```

3.1 RESULT = PROJECT Location from Employees

The PROJECT operation is implemented by `SELECT <columns> FROM <table>` command in most SQL implementations. So, the result of this operation is:

```

> pd.read_sql("SELECT Location FROM Employees", conn)

```

```

Out[18]:   Location
0         5
1         3
2         5

```

3.2 RESULT = SELECT from Employees where Location = 5

The SELECT operation is to select the rows, and is implemented by `SELECT * FROM <table> WHERE <condition>` in most SQL implementations. So, the result of this operation is:



```
> pd.read_sql("SELECT * FROM Employees WHERE Location = 5", conn)
```

```
Out[19]:
```

	Name	Email_address	Location
0	John	John.john@abxz.com	5
1	Philip	philipx@abxz.com	5

3.3 RESULT = PROJECT Floor_# from Department

Same as the first operation.



```
> pd.read_sql("SELECT Floor_Number FROM Department", conn)
```

```
Out[9]:
```

	Floor_Number
0	4
1	5

3.4 RESULT = JOIN Employees and Department where Employees.Location = Department.Floor_Number

The JOIN operation is to join the columns of two tables with a given condition. In SQL, it's often implemented as `SELECT * FROM <table1> JOIN <table2> WHERE <condition>`. So the result is:



```
pd.read_sql("""SELECT *
              FROM Employees
              JOIN Department
              WHERE Employees.Location = Department.Floor_Number
              """, conn)
```

```
Out[22]:
```

	Name	Email_address	Location	Manager_Name	Floor_Number
0	John	John.john@abxz.com	5	David	5
1	Philip	philipx@abxz.com	5	David	5

Be a good citizen and close the connection before we leave.



```
? conn.close()
```

References

Brookshear, J. G. (2011), *Computer science: an overview*, Paul Muljadi.

- McKinney, W. (2010), Data structures for statistical computing in Python, *in* S. J. van der Walt & K. J. Millman, eds, ‘proceedings of the 9th Python in Science Conference’, Austin, Texas.
- Owens, M. & Allen, G. (2006), *The definitive guide to SQLite*, Vol. 1, Springer.
- Pérez, F. & Granger, B. E. (2007), ‘IPython: a System for Interactive Scientific Computing’, *Computing in Science & Engineering* **9**(3), 21–29. URL: <http://ipython.org>.
- Wikipedia (2014a), ‘Database administrator — wikipedia, the free encyclopedia’. [Online; accessed 16-November-2014].
URL: http://en.wikipedia.org/w/index.php?title=Database_administrator&oldid=629081597
- Wikipedia (2014b), ‘Partition (database) — wikipedia, the free encyclopedia’. [Online; accessed 17-November-2014].
URL: [http://en.wikipedia.org/w/index.php?title=Partition_\(database\)&oldid=616221891](http://en.wikipedia.org/w/index.php?title=Partition_(database)&oldid=616221891)
- Wikipedia (2014c), ‘Projection (relational algebra) — wikipedia, the free encyclopedia’. [Online; accessed 17-November-2014].
URL: [http://en.wikipedia.org/w/index.php?title=Projection_\(relational_algebra\)&oldid=618853777](http://en.wikipedia.org/w/index.php?title=Projection_(relational_algebra)&oldid=618853777)