Using SQL with Python for Data Analysis

Ben Winjum
OARC

November 5, 2021

Overview

- Some basics about SQL and Python
    - And a brief SQL crash course
- My goal is to give you enough exposure to SQL and Python that you can be in a position to learn more
- The bulk of our material will be executing and discussing Python code in the notebooks I've shared
    - You are welcome to follow along on your own, but it won't strictly be necessary

SQL vs Python as languages

- Python is an imperative language:
  - Like C++, Java, Fortran….
  - You give the computer specific instructions about how to execute your algorithmic desires
  - “Computer, here is how I want you to change your state …”
- SQL is a declarative query language:
  - “Computer, I want data that meets the following criteria …”
  - The RDBMS can store the data how it wants, and its query planner can figure out how to get the data

SQL	Python
Query language	Imperative language
Used to get information out of relational database systems	Used for backend web development, data analysis, scientific computing, ML/AI, ….
Databases are useful when dealing with ● lots of data ● frequent updates to data ● simultaneous changes to data ● shared data among a lot of people ● rapid queries without much analysis	Python’s emphasis is on tackling computational problems, preferably with simple and readable code -- it can be easy to learn, but it's also a powerful language with many libraries that enhance its ability to efficiently tackle a wide range of problems

SQL Queries, the ultra-short version

Students

studentId	Name	classId
1	Dora	1
2	Daniel	2
3	Mamdooh	1
4	Lana	2
5	Ben	4

Classes

classId	Title
1	Film002
2	Econ243
3	Phys100

studentId	Name	classId
1	Dora	1
2	Daniel	2
3	Mamdooh	1
4	Lana	2
5	Ben	4

SQL Essentials

```
SELECT *
FROM Students;
```

studentId	Name	classId
1	Dora	1
2	Daniel	2
3	Mamdooh	1
4	Lana	2
5	Ben	4

SQL Essentials

```
SELECT *
FROM Students;
```

--Or similarly--

SELECT column1, column2, AGG_FUNC(*column_or_expression*), ...
FROM Students;

SQL Essentials

studentId	Name	classId
1	Dora	1
2	Daniel	2
3	Mamdooh	1
4	Lana	2
5	Ben	4

```
SELECT *
FROM Students
WHERE classId = 2;
```

studentId	Name	classId
2	Daniel	2
4	Lana	2

studentId	Name	classId
1	Dora	1
2	Daniel	2
3	Mamdooh	1
4	Lana	2
5	Ben	4

SQL Essentials

```
SELECT classId, count(*)
FROM Students
WHERE studentId < 6
GROUP BY classId;
```

classId	count(*)
1	2
2	2
4	1

studentId	Name	classId
1	Dora	1
2	Daniel	2
3	Mamdooh	1
4	Lana	2
5	Ben	4

SQL Essentials

```
SELECT classId, count(*)

FROM Students

WHERE studentId < 6

GROUP BY classId

HAVING count(*) < 2;
```

classId	count(*)
4	1

studentId	Name	classId
1	Dora	1
2	Daniel	2
3	Mamdooh	1
4	Lana	2
5	Ben	4

SQL Essentials

```
SELECT classId, count(*)
FROM Students
WHERE studentId < 6
GROUP BY classId
HAVING classId < 3
ORDER BY classId DESC;
```

classId	count(*)
2	2
1	2

studentId	Name	classId
1	Dora	1
2	Daniel	2
3	Mamdooh	1
4	Lana	2
5	Ben	4

SQL Essentials

```
SELECT classId, count(*)
FROM Students
WHERE studentId < 6
GROUP BY classId
HAVING classId < 3
ORDER BY classId DESC
LIMIT 1;
```

classId	count(*)
2	2

studentId	Name	classId
1	Dora	1
2	Daniel	2
3	Mamdooh	1
4	Lana	2
5	Ben	4

SQL Essentials

classId	Title
1	Film002
2	Econ243
3	Phys100

SELECT classId, count(*), title

FROM Students s

INNER JOIN Classes c

ON s.classId = c.classId

WHERE studentId < 6

GROUP BY classId

HAVING classId < 3

ORDER BY classId DESC

LIMIT 1;

classId	count(*)	title
2	2	Econ243

SQL Essentials

```
SELECT columnname, AGG_FUNC(column_or_expression),...
FROM mytable
INNER JOIN another_table
ON mytable.classId = another_table.classId
WHERE constraint_expression
GROUP BY column
HAVING constraint_expression
ORDER BY column ASC/DESC
LIMIT count OFFSET COUNT;
```

Varieties of RDBMs

- We'll been using SQLite, but there are many other options
- SQLite is:
  - Very light-weight
  - Unique in its operation without a database server (so doesn't require configuration)
  - A database is stored entirely in a file

Using SQL with Python

- SQL is geared towards relations between data,
- Python uses objects (and is an object-oriented language)

- So, if you explore SQL+Python further, you’ll also run into SQLAlchemy
    - Python library that provides an object relational mapper (ORM)
    - Maps databases (tables, etc.) to Python objects for easier interaction
    - Lets you use models consistently across engines and can be configured to use any of SQLite, MySQL, PostgreSQL, etc. underneath the hood

Using SQL with Python

- SQLite, MySQL, PostgreSQL, etc are database storage engines that are used to store and retrieve structured data from files
- AND, many can be readily integrated with Python with an appropriate library
    - Python already has built-in support for SQLite: “import sqlite3”
    - There are also readily available packages for working with other RDBMs:
        - “psycopg2” for PostgreSQL
        - “pymysql” for MySQL,
        - “cx_Oracle” for Oracle Database …

Using SQL with Python

- The basic method that Python uses to interface with databases is standardized (Python DB-API), so the flow of code looks similar regardless of the database model used

Using SQL with Python

- Import the relevant library
    - “import sqlite3”

Using SQL with Python

- Import the relevant library
  - “import sqlite3”
- Create a connection object that represents the database
  - “conn = sqlite3.connect(‘databasefile’)”

Using SQL with Python

- Import the relevant library
    - "import sqlite3"
- Create a connection object that represents the database
    - "conn = sqlite3.connect('databasefile')"
- Get a cursor object
    - "cur = conn.cursor()"

Using SQL with Python

- Import the relevant library
    - “import sqlite3”
- Create a connection object that represents the database
    - “conn = sqlite3.connect(‘databasefile’)”
- Get a cursor object
    - “cur = conn.cursor()”
- Use this cursor to execute SQL commands
    - Like ”cur.execute(query)” or “cur.fetchall()”

Using SQL with Python

- Import the relevant library
    - “import sqlite3”
- Create a connection object that represents the database
    - “conn = sqlite3.connect(‘databasefile’)”
- Get a cursor object
    - “cur = conn.cursor()”
- Use this cursor to execute SQL commands
    - Like ”cur.execute(query)” or “cur.fetchall()”
- The executed queries don’t write to the database until an explicit command is given to commit the transaction (“conn.commit()”)

Using SQL with Python

- Import the relevant library
    - “import sqlite3”
- Create a connection object that represents the database
    - “conn = sqlite3.connect(‘databasefile’)”
- Get a cursor object
    - “cur = conn.cursor()”
- Use this cursor to execute SQL commands
    - Like ”cur.execute(query)” or “cur.fetchall()”
- The executed queries don’t write to the database until an explicit command is given to commit the transaction (“conn.commit()”)
- Close the cursor and connection objects

Let’s start working with SQL!

Go to https://github.com/benjum/oarc-fall21-sql-python

And click on the the JupyterHub (or Binder) link

Any Future Questions:

Feel free to email me – bwinjum@oarc.ucla.edu