SQL-Python

Sean Sall

May 12th, 2016

Objectives:

Today's objectives:

- Learn how to connect and run `Postgres` queries from Python
- Understand `psycopg2`'s cursors, specifically executes and commits
- Learn how to generate dynamic queries from within Python

Agenda

- SQL-Python motivation
- `psycopg2` introduction and workflow
  - Creating databases
  - Executing queries (static and dynamic)

SQL-Python Motivation

- Why write SQL queries from within Python?
  - Allows for the combination of data sources all in one place (e.g. you can use Python to pull data from other databases as well)
  - Allows for use of all our Python tools when working with the data (dataframes, machine learning models, etc.)
  - Allows for more easy dynamic query generation and hence automations

psycopg2

- psycopg is the Python library that we'll use to interface to a Postgres database from within Python
- There are Python libraries to connect to almost any database that you might want to:
  - MySql : `mysql-connector-python`
  - SQLite : `sqlite`
  - MongoDB : `pymongo`

psycopg2 Workflow

- There are five general steps to a pyscopg2 workflow:
  1. Open a connection
  2. Create a `cursor` object
  3. Use the `cursor` to `execute` SQL queries
  4. `Commit` SQL actions
  5. Close the cursor and connection

Open a Connection

```
import pyscopg2

conn = psycopg2.connect(dbname='my_playground',
                        user='sallamander', host='localhost')
```

- The `host` can be used to connect to a remote database as well
- Sometimes we have to put in the `password` or `port` arguments as well

Create a `cursor` Object

```
cur = conn.cursor()
```

- A **cursor** is a control structure that enables traversal over the records in a database
  - Executes queries to fetch data
  - Handles transactions with our SQL database
- When results are returned from a cursor object, they are returned as a generator (e.g. it gives back the results lazily)
- Furthermore, each result in the result set can only be accessed **once** (if we want it again we have to re-run the query)

Use the cursor to execute SQL queries

```
query = '''SELECT name
           FROM users
           INNER JOIN disney_chars
               ON users.fav_disney_char = disney_chars.id
            WHERE disney_chars.prince = 'true';'''
cur.execute(query)
```

- The cursor object (cur) will now hold the results

Getting results from the `cursor`

- There are a number of ways to grab results from the `cursor`:
  - `cur.fetchone()` - Returns the next result
  - `cur.next()` - Returns the next result
  - `cur.fetchmany(n)` - Returns the next `n` results
  - `cur.fetchall()` - Returns all results in the result set
  - `for res in cur:` - Iterates over all results in the `cursor`

Commit the Results

- Data changes are not actually stored until you **commit** them
  - This is only important if you are creating a database/datatable, or altering the data in an existing database/datable

```
query='''ALTER TABLE disney_chars
         RENAME mulan TO the_best;'''
cur.execute(query)
conn.commit()
```

- The column's name is **not changed in the `db`** until we **issue the `commit`**
  - It is, however, **changed on the `connection` and `cursor`** as soon as we **issue the `execute`**

Rollback

- If you make a mistake on a query, you need to use the rollback function to restart the transaction

```
conn.rollback()
```

Close the connection

- Don't forget to do this!!

```
cur.close()
conn.close()
```

- Closing the `cursor` is technically optional because closing the `connection` closes all `cursor` objects, but it is good practice to close both

A Short Note

- Anything executed through the `.query` method on the `cursor` is done so as a **temporary transaction**. Since Postgres doesn't have these at the database level, we have to specify an additional attribute on the `connection` before trying to perform database level operations (create/dropping databases)

```
conn = psycopg2.connect(dbname='my_playground',
                        user='sallamander', host='localhost')
conn.autocommit = True
```

Dynamic Queries

- psycopg2 gives us the ability to create dynamic queries, where we can insert certain values into our queries on the fly.

Dynamic Queries - The Wrong Way

- **THE WRONG WAY** to write a dynamic query:

```
unsafe_query = '''SELECT * FROM users
                  WHERE name =
                '''' + name_var

other_unsafe_query = '''SELECT * FROM
                       users WHERE name = %s
                     ''' % name_var
```

What happens if somebody inputs a `name_var` equal to " 'Sean'; DROP TABLE users;"?

Something like this is referred to as **SQL INJECTION**.

Dynamic Queries - The Right Way

- The **RIGHT WAY** to write a dynamic query is to use the `.execute()` method on our `cursor()` object, passing the dynamic part as the second argument:

```
cursor.execute('''SELECT * FROM
                  users WHERE name = %s''', (name_var, ))
```

- This ensures that the variables you are inserting are kept as the same variable type. If we tried to perform **SQL INJECTION** using " 'Sean'; DROP TABLE users;", using the `.execute()` method means that we look for a `name` exactly equal to this string (and we don't end up finding one, nor deleting the data table).