Natalie Sprint #2 Review

Connecting a Data Source and Database using Python and PostgreSQL

devleague

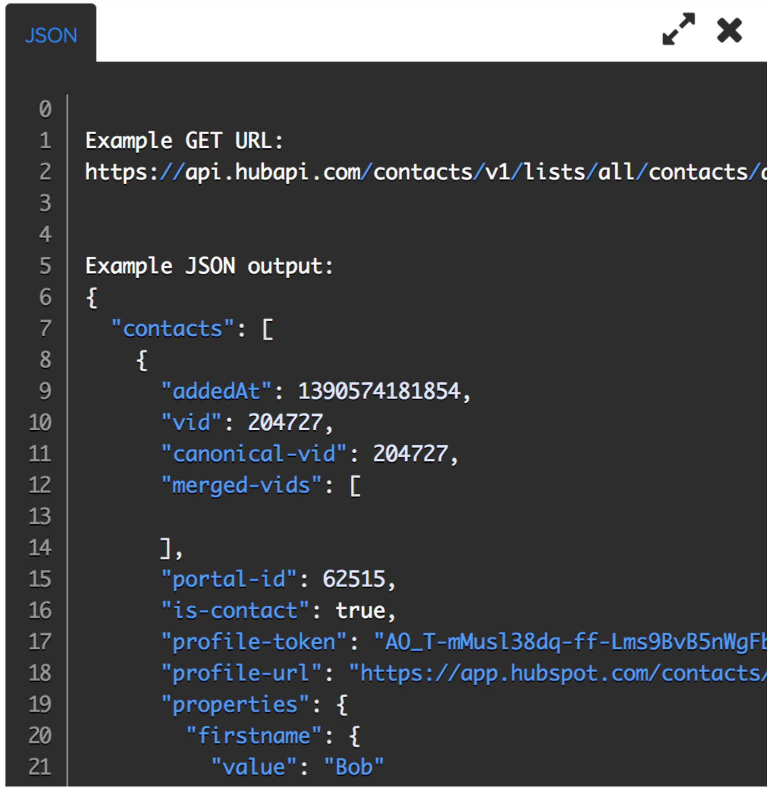
Project Overview

- Connect to HubSpot's API and pull in all contact data.
- Parse data in extract only certain fields to make new data structure.
- Create PostgreSQL database with data model for business needs.
- Use new data structure to insert values into existing table in a PostgreSQL database.

devleague

Accessing HubSpot’s API

- Hubspot’s API documents were not extremely helpful, but they did provide examples of what an api call would look like.
- They have an api called ’get all contacts’ which gives you on huge json object.
- The object contained nested lists and dictionaries.



```
Example GET URL:
https://api.hubapi.com/contacts/v1/lists/all/contacts/c


Example JSON output:
{
  "contacts": [
    {
      "addedAt": 1390574181854,
      "vid": 204727,
      "canonical-vid": 204727,
      "merged-vids": [

      ],
      "portal-id": 62515,
      "is-contact": true,
      "profile-token": "AO_T-mMusl38dq-ff-Lms9BvB5nWgFb
      "profile-url": "https://app.hubspot.com/contacts/
      "properties": {
        "firstname": {
          "value": "Bob"
```

Python Script for HubSpot API

- Python libraries used for this script:
    - pprint
    - urllib.request
    - Json

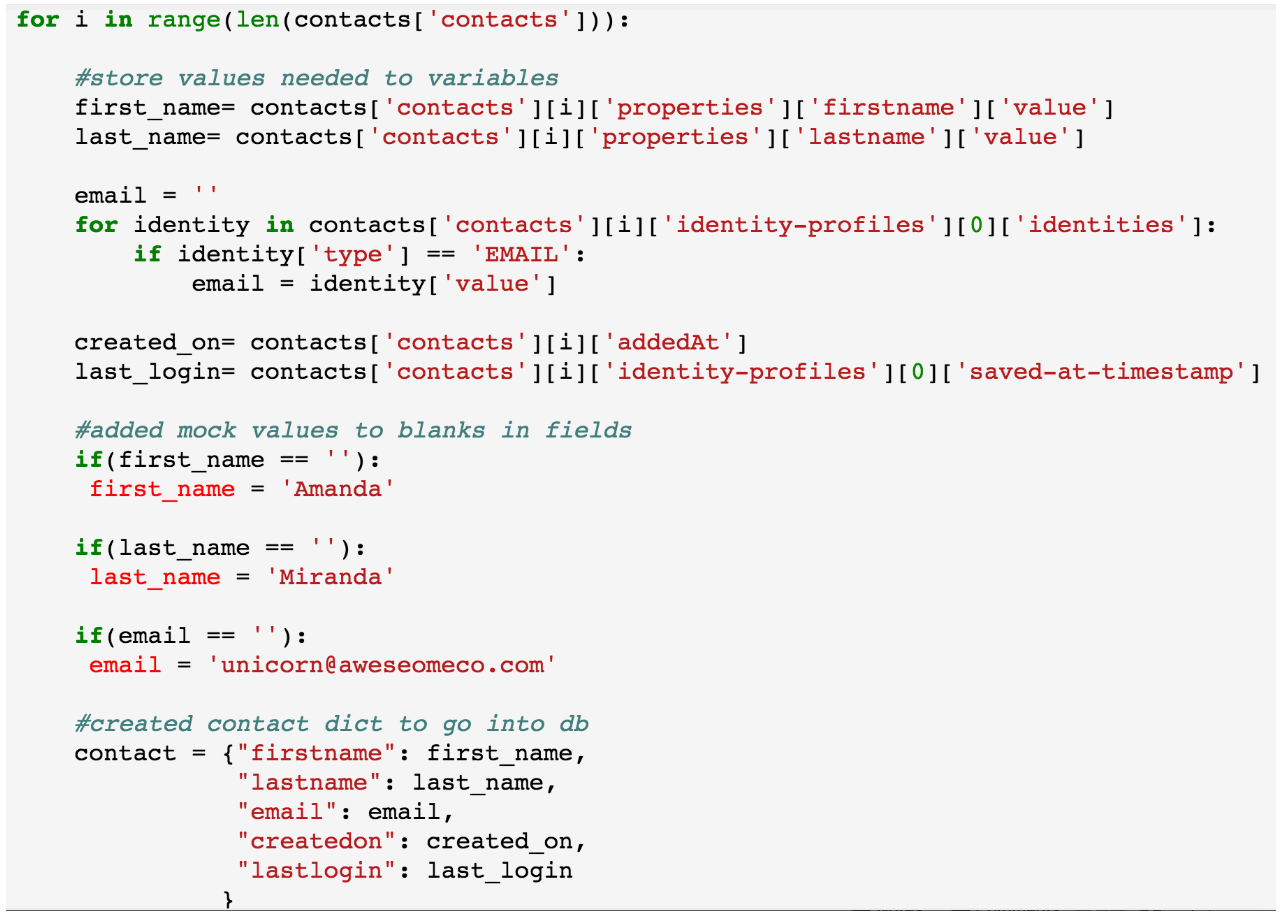
- Used urllib.request to open and read api url
- Used json.loads to load response into json
- Invoked the function to load the list

```python
from pprint import pprint
import urllib.request, json

# Store api key value into variable
APIKEY_VALUE = "demo"

# concat query string with api key
APIKEY = "?hapikey=" + APIKEY_VALUE

# hs api end point stored to a variable
HS_API_URL = "http://api.hubapi.com"

def get_contacts():
    # builds the correct url
    xurl = "/contacts/v1/lists/all/contacts/all"
    url = HS_API_URL + xurl + APIKEY
    # Now we use urllib to open the url and read it
    response = urllib.request.urlopen(url).read()
    #loads to json obj to all_contacts variable
    all_contacts = json.loads(response)
    #return the contact data
    return all_contacts

# calls the function stores to variable
contacts = get_contacts()

# pretty print just the first name of a contact
pprint(contacts)
```

devleague

Looping through keys in Dictionary

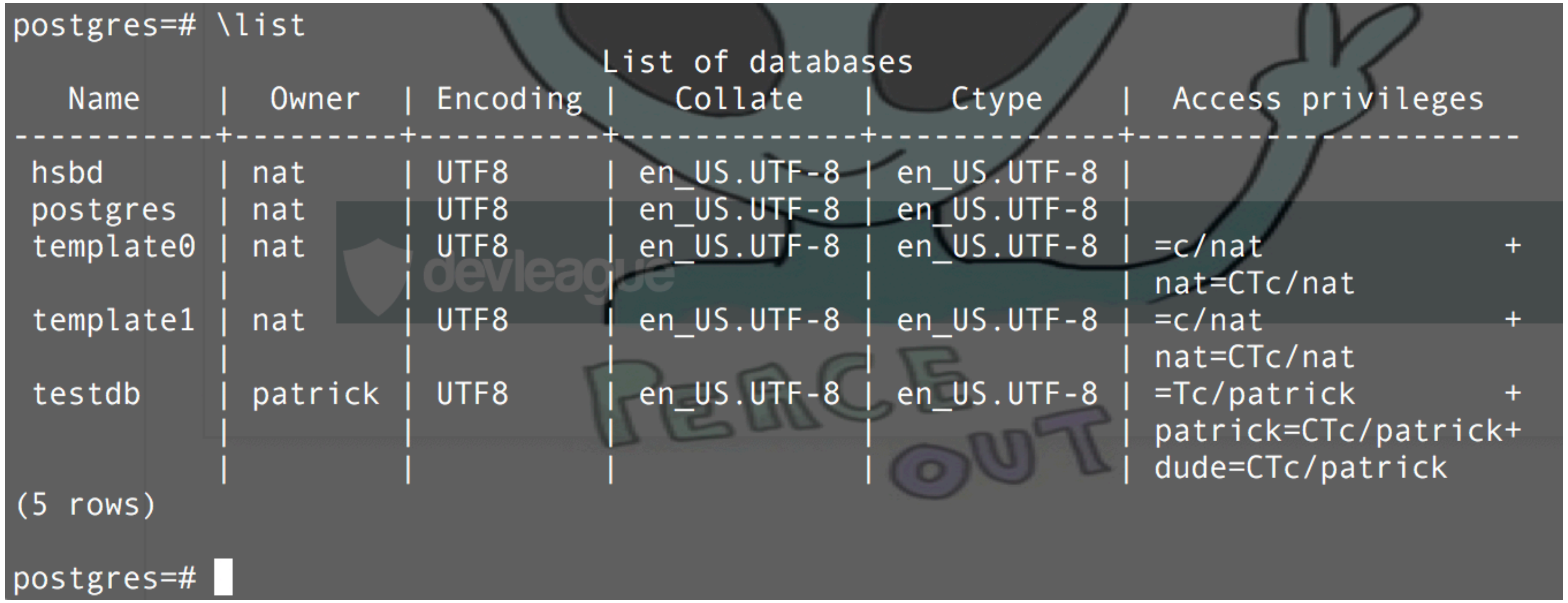
- Values needed were in nested lists and dictionaries within the json object.
- Had to create for loop to go through the range of keys in the whole contact list
- Wrote logic that extracted data and cleaned it
- Formatted values extracted from json into new cleaned dictionary.



```
for i in range(len(contacts['contacts'])):

    #store values needed to variables
    first_name= contacts['contacts'][i]['properties']['firstname']['value']
    last_name= contacts['contacts'][i]['properties']['lastname']['value']

    email = ''
    for identity in contacts['contacts'][i]['identity-profiles'][0]['identities']:
        if identity['type'] == 'EMAIL':
            email = identity['value']

    created_on= contacts['contacts'][i]['addedAt']
    last_login= contacts['contacts'][i]['identity-profiles'][0]['saved-at-timestamp']

    #added mock values to blanks in fields
    if(first_name == ''):
     first_name = 'Amanda'

    if(last_name == ''):
     last_name = 'Miranda'

    if(email == ''):
     email = 'unicorn@aweseomeco.com'

    #created contact dict to go into db
    contact = {"firstname": first_name,
               "lastname": last_name,
               "email": email,
               "createdon": created_on,
               "lastlogin": last_login
              }
```

devleague

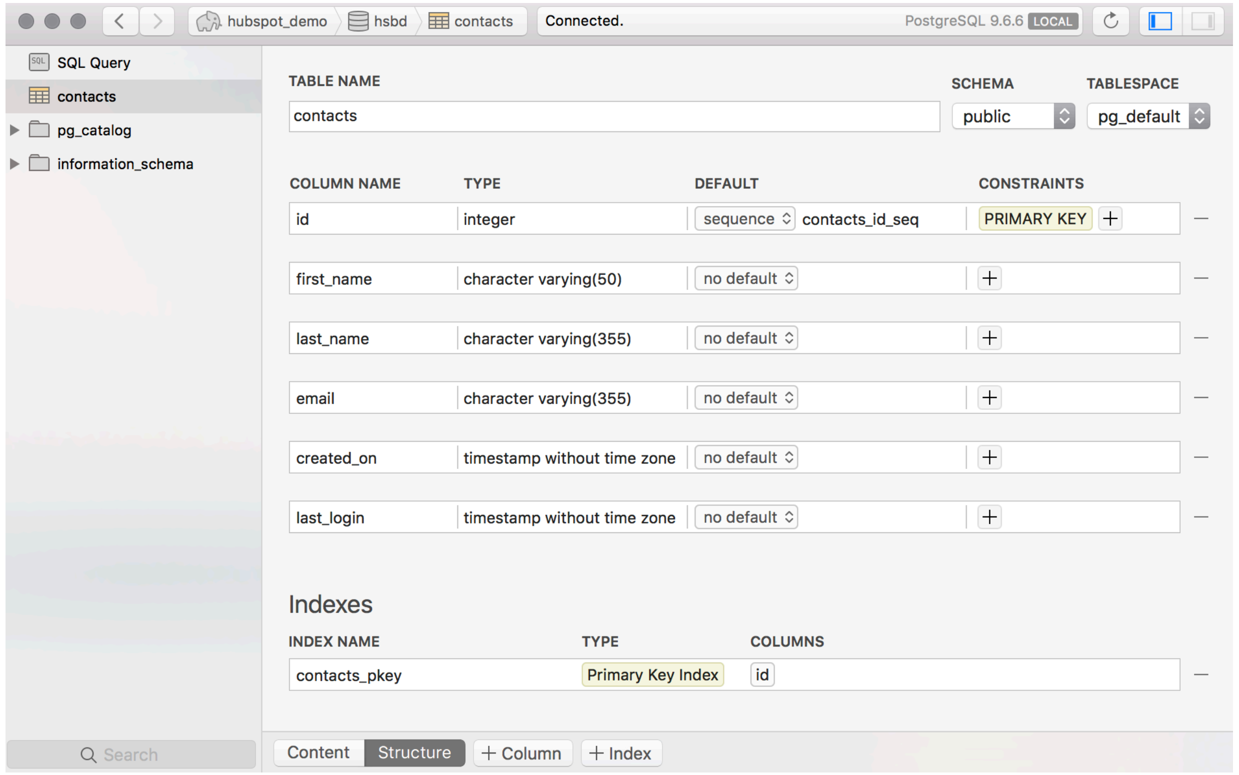
Setting up PostgreSQL Database



```
postgres=# \list
                                 List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 hsbd      | nat      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 postgres  | nat      | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
 template0 | nat      | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/nat               +
           |          |          |             |             | nat=CTc/nat
 template1 | nat      | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =c/nat               +
           |          |          |             |             | nat=CTc/nat
 testdb    | patrick  | UTF8     | en_US.UTF-8 | en_US.UTF-8 | =Tc/patrick          +
           |          |          |             |             | patrick=CTc/patrick+
           |          |          |             |             | dude=CTc/patrick
(5 rows)

postgres=# 
```

Setting up Postico GUI

- Love POSTICO!!!
- Easy to install and use
- Easy to set up data model and tables
- Allows same features as creating it from the command line



hubspot_demo hsbd contacts Connected. PostgreSQL 9.6.6 LOCAL

TABLE NAME	SCHEMA	TABLESPACE
contacts	public	pg_default

COLUMN NAME	TYPE	DEFAULT	CONSTRAINTS
id	integer	sequence contacts_id_seq	PRIMARY KEY
first_name	character varying(50)	no default	
last_name	character varying(355)	no default	
email	character varying(355)	no default	
created_on	timestamp without time zone	no default	
last_login	timestamp without time zone	no default	

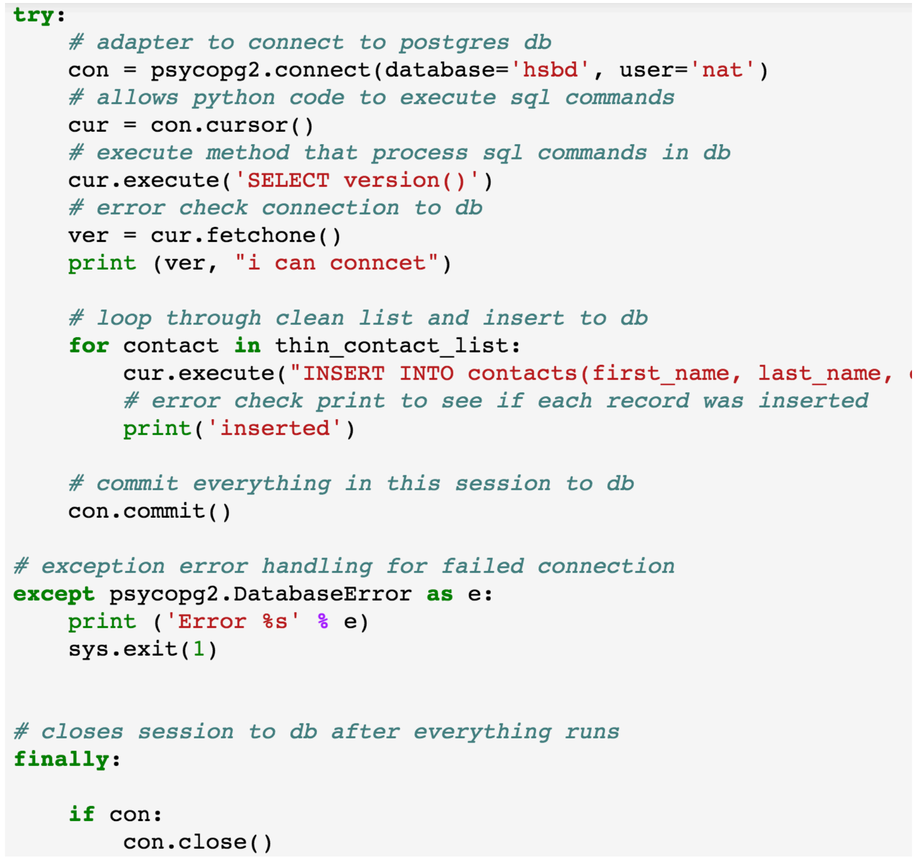
Indexes

INDEX NAME	TYPE	COLUMNS
contacts_pkey	Primary Key Index	id

devleague

Connecting to Database with Python

- Libraries Used:
  - psycopg2 to connect to PostgreSQL
  - sys to stop python script and exit database session
- Used the **execute** method to run SQL commands.
- Really useful library - was very easy to use.

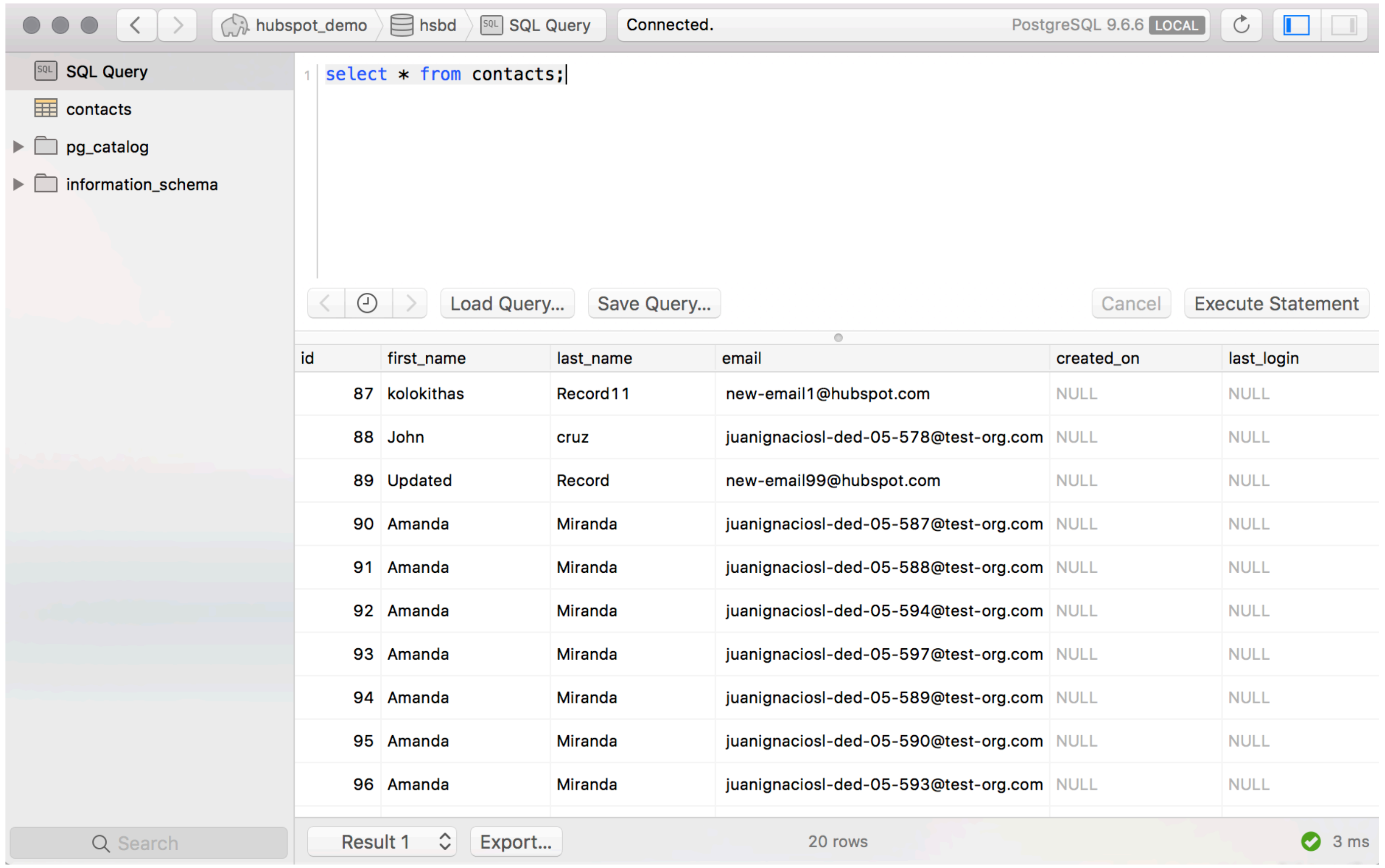


```python
try:
    # adapter to connect to postgres db
    con = psycopg2.connect(database='hsbd', user='nat')
    # allows python code to execute sql commands
    cur = con.cursor()
    # execute method that process sql commands in db
    cur.execute('SELECT version()')
    # error check connection to db
    ver = cur.fetchone()
    print (ver, "i can conncet")

    # loop through clean list and insert to db
    for contact in thin_contact_list:
        cur.execute("INSERT INTO contacts(first_name, last_name, 
        # error check print to see if each record was inserted
        print('inserted')

    # commit everything in this session to db
    con.commit()

# exception error handling for failed connection
except psycopg2.DatabaseError as e:
    print ('Error %s' % e)
    sys.exit(1)


# closes session to db after everything runs
finally:

    if con:
        con.close()
```

Running Scripts in CLI VS. Jupyter

- In Jupyter when you have multiple cells, as long as you’re in the same notebook it treats it like one big file.
- When you run the files in the command line you have to create **Modules**
- To connect files, all you have to do is connect files as you would import a library.

Results !!!!! ☺



```
select * from contacts;
```

id	first_name	last_name	email	created_on	last_login
87	kolokithas	Record11	new-email1@hubspot.com	NULL	NULL
88	John	cruz	juanignaciosl-ded-05-578@test-org.com	NULL	NULL
89	Updated	Record	new-email99@hubspot.com	NULL	NULL
90	Amanda	Miranda	juanignaciosl-ded-05-587@test-org.com	NULL	NULL
91	Amanda	Miranda	juanignaciosl-ded-05-588@test-org.com	NULL	NULL
92	Amanda	Miranda	juanignaciosl-ded-05-594@test-org.com	NULL	NULL
93	Amanda	Miranda	juanignaciosl-ded-05-597@test-org.com	NULL	NULL
94	Amanda	Miranda	juanignaciosl-ded-05-589@test-org.com	NULL	NULL
95	Amanda	Miranda	juanignaciosl-ded-05-590@test-org.com	NULL	NULL
96	Amanda	Miranda	juanignaciosl-ded-05-593@test-org.com	NULL	NULL

devleague

Thank you

devleague