Aileen Novero
CS 8674 – Independent Master's Project
Summer Term 2016
Review of Project
Supervisor: Dr. Ian Gorton, Professor and Director of Computer Science, Seattle Campus

Create, Host, Deploy in Python-ese

Creating a mySQL database web application
Hosted by Flask & Publicly Deployed on Amazon Web Services

Introduction

As a neophyte utilizing the numerous packages available to the ever-increasing industry used Python language, I pursued the following project in order to have the opportunity to explore and implement tools and skills prized by the Computer Science industry, in particular Data Science.

As a note there are currently two versions of the requested Flaskapp for the Scientific Abstracts Database. One runs on a local machine, one is publicly deployed on Amazon Web Services. Each are available at public Github repositories for code searching, improvements, and expansion.

Local : https://github.com/noveroa/DataBases/

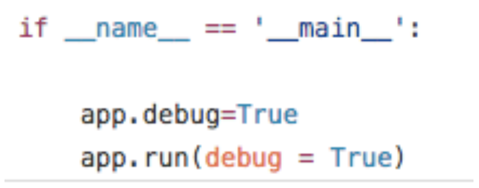
AWS : https://github.com/noveroa/DataBaseAWS

Objective:
To build a Web-based data capture, storage and analysis system for academic conference papers. The initial data set was based on a collection of text files from three software architecture conferences over a span of the years 1999 to 2014. Each included paper titles, authors, abstracts, affiliations and keywords. The aim is to provide a user with a collection of analysis methods supported by Web-based visualization to explore the abstracts in interesting ways.

Learning Objectives:
The project was originally designed to include three main learning objectives. Two can be coalesced into the second topic, with the third, the inclusion of the public deployment added.

- Design a database for storing textual data and associated metadata:
    - Python Relational Database
        - Python, Python Pandas, sqlite3
        - Cleaning and parsing complex text files
        - Creation of a database (with interest in relational)
- Implement a Web-based system
    - Enable a user to submit queries and search based on specific attributes of the data set (i.e. year(s) of publication, author, institution, topics)
    - Visualize the results using Web-based visualization
        - Python visualization packages : maplotlib, Seaborn, wordcloud
        - Hosting the database in a web-application useable on local browsers
    - Hosting the database in a web-application useable on local browsers
        - Flask – Hosted Web Application
        - Python, Flask, jQuery, apache, Bootstrap

- Implement and Deploy a public server for the web-based application.
    - Amazon Web Services
        - Deploying the Flask application useable to customers
        - AWS, Ubuntu, apache, mod-wsgi

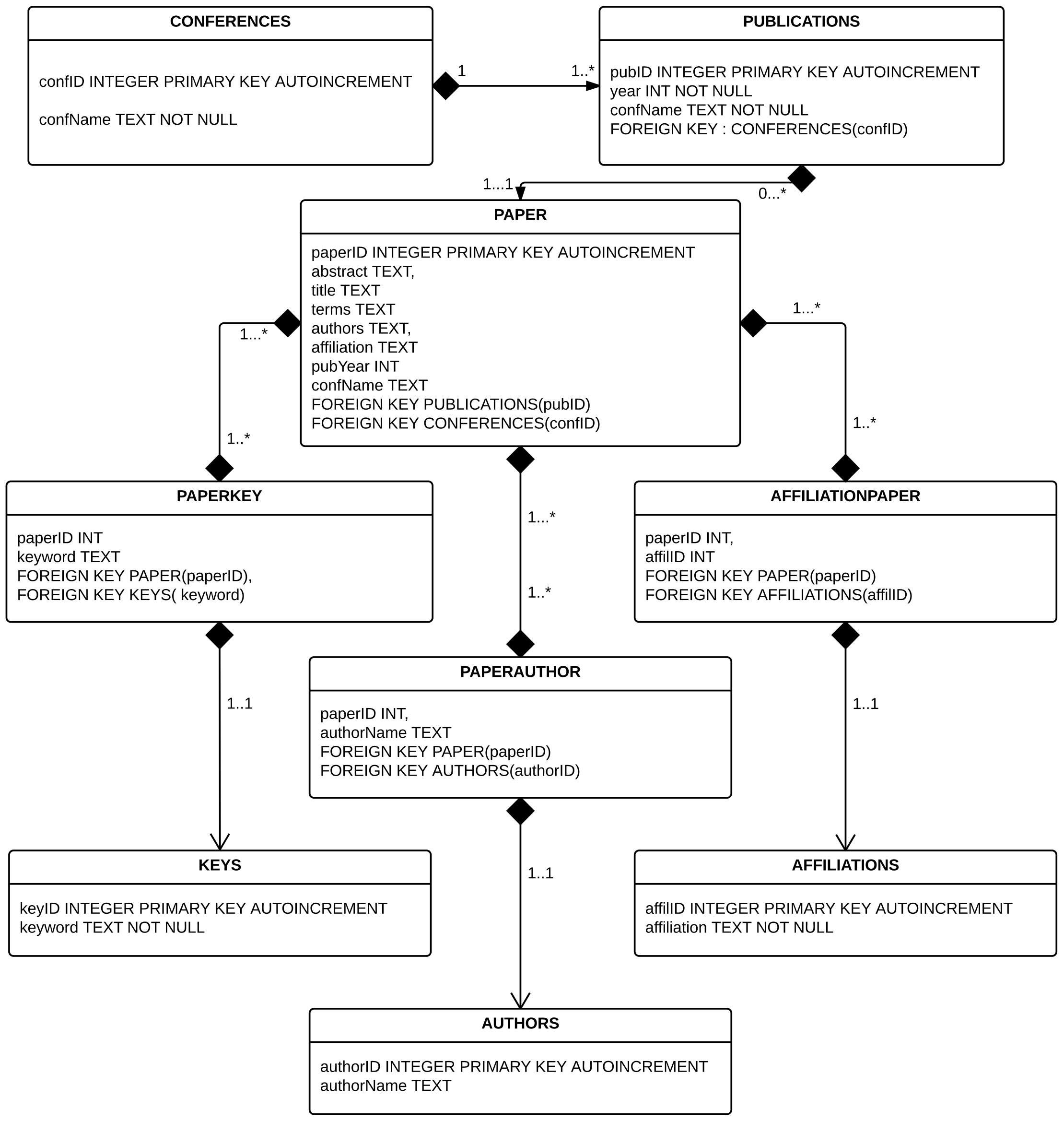


```
if __name__ == '__main__':

    app.debug=True
    app.run(debug = True)
```

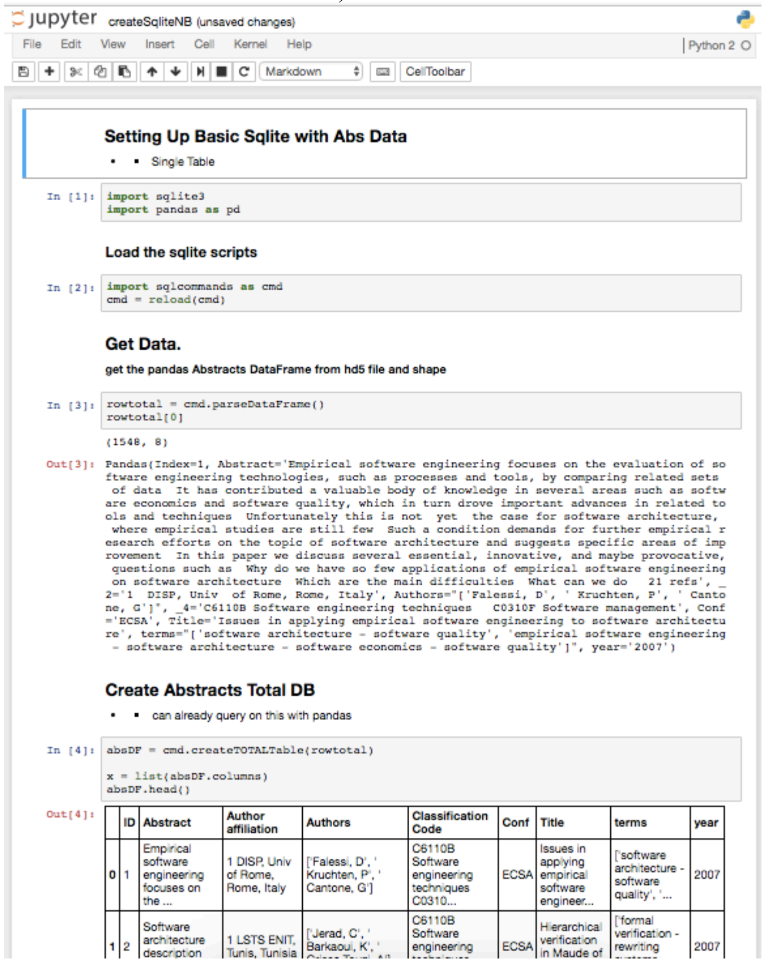
The Database

The database was created implementing the packages for Python utilizing Python Pandas DataFrames and sqlite3, a syntax similar to MySql. Sqlite was chosen due to its inherent compatibility with python and pandas. There are limits to utilizing sqlite (including no full joins, foreign key issues).

The database schema:


CONFERENCES
confID INTEGER PRIMARY KEY AUTOINCREMENT
confName TEXT NOT NULL
PUBLICATIONS
pubID INTEGER PRIMARY KEY AUTOINCREMENT
year INT NOT NULL
confName TEXT NOT NULL
FOREIGN KEY : CONFERENCES(confID)
1
1..*
1...1
0...*
PAPER
paperID INTEGER PRIMARY KEY AUTOINCREMENT
abstract TEXT,
title TEXT
terms TEXT
authors TEXT,
affiliation TEXT
pubYear INT
confName TEXT
FOREIGN KEY PUBLICATIONS(pubID)
FOREIGN KEY CONFERENCES(confID)
1...*
1..*
1...*
1..*
1...*
1..*
PAPERKEY
paperID INT
keyword TEXT
FOREIGN KEY PAPER(paperID),
FOREIGN KEY KEYS(keyword)
AFFILIATIONPAPER
paperID INT,
affilID INT
FOREIGN KEY PAPER(paperID)
FOREIGN KEY AFFILIATIONS(affilID)
PAPERAUTHOR
paperID INT,
authorName TEXT
FOREIGN KEY PAPER(paperID)
FOREIGN KEY AUTHORS(authorID)
1..1
1..1
1..1
KEYS
keyID INTEGER PRIMARY KEY AUTOINCREMENT
keyword TEXT NOT NULL
AFFILIATIONS
affilID INTEGER PRIMARY KEY AUTOINCREMENT
affiliation TEXT NOT NULL
AUTHORS
authorID INTEGER PRIMARY KEY AUTOINCREMENT
authorName TEXT

I had inherently begun this process determined to make a relational database using python and sqlite3. Python being a language used more and more often, rather than Java. Also, environments can be created using Anaconda and conda – and maintain updated packages as necessary. Further, Jupyter notebooks are a great learning and sharing tool for testing codes without the command line or writing scripts first. It supports nice markup to also explain what you are working on.
Sample Notebook (with a Pandas DataFrame included)


jupyter
createSqliteNB (unsaved changes)
File Edit View Insert Cell Kernel Help
Python 2
Markdown
CellToolbar

Setting Up Basic Sqlite with Abs Data

- Single Table

```
In [1]: import sqlite3
        import pandas as pd
```

Load the sqlite scripts

```
In [2]: import sqlcommands as cmd
        cmd = reload(cmd)
```

Get Data.

get the pandas Abstracts DataFrame from hd5 file and shape

```
In [3]: rowtotal = cmd.parseDataFrame()
        rowtotal[0]

        (1548, 8)

Out[3]: Pandas(Index=1, Abstract='Empirical software engineering focuses on the evaluation of so
        ftware engineering technologies, such as processes and tools, by comparing related sets
         of data  It has contributed a valuable body of knowledge in several areas such as softw
        are economics and software quality, which in turn drove important advances in related to
        ols and techniques  Unfortunately this is not  yet  the case for software architecture,
         where empirical studies are still few  Such a condition demands for further empirical r
        esearch efforts on the topic of software architecture and suggests specific areas of imp
        rovement  In this paper we discuss several essential, innovative, and maybe provocative,
         questions such as  Why do we have so few applications of empirical software engineering
         on software architecture  Which are the main difficulties  What can we do   21 refs', _
        2='1  DISP, Univ  of Rome, Rome, Italy', Authors="['Falessi, D', ' Kruchten, P', ' Canto
        ne, G']", _4='C6110B Software engineering techniques   C0310F Software management', Conf
        ='ECSA', Title='Issues in applying empirical software engineering to software architectu
        re', terms="['software architecture - software quality', 'empirical software engineering
         - software architecture - software economics - software quality']", year='2007')
```

Create Abstracts Total DB

- can already query on this with pandas

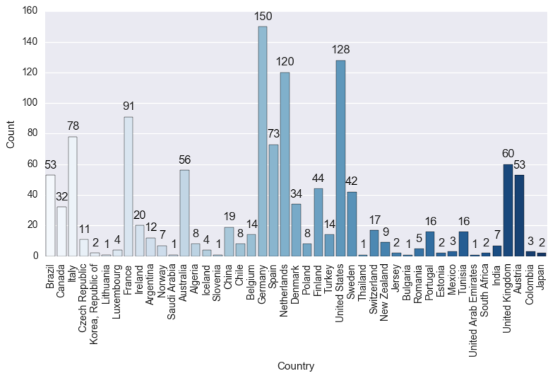
```
In [4]: absDF = cmd.createTOTALTable(rowtotal)

        x = list(absDF.columns)
        absDF.head()
```

Out[4]:

	ID	Abstract	Author affiliation	Authors	Classification Code	Conf	Title	terms	year
0	1	Empirical software engineering focuses on the ...	1 DISP, Univ of Rome, Rome, Italy	['Falessi, D', ' Kruchten, P', ' Cantone, G']	C6110B Software engineering techniques C0310...	ECSA	Issues in applying empirical software engineer...	['software architecture - software quality', '...	2007
1	2	Software architecture description	1 LSTS ENIT, Tunis, Tunisia	['Jerad, C', ' Barkaoui, K', '	C6110B Software engineering	ECSA	Hierarchical verification in Maude of	['formal verification - rewriting	2007

Concerning the relational database aspect, although, SQLAlchemy can also be utilized for more manageable migration and foreign key manipulation. I did ultimately start and continued to use sqlite3 alone for the small database. It is interesting to note, though, that Python DataFrames are also an easy and intuitive way to build a database as well, though foreign keys are not supported. The limits here seem to be the amount of data that can be handled and creating well structured normal forms. Yet, many of the manipulations that are done via the creation of normal form tables could be done with a creation of a single DataFrame. Merging, selecting, combining, aggregating, and querying are quite easy. Further one can easily drop duplicates and find counts of unique entries per column r selected columns. Again, speed and memory may be impacted by this approach. Thus, the sql database was used. As the project moved closer to the final stages, more sql-like statements, rather than pandas, were used. I had thought that future collaborators may find it more useful – in particular if the database expands. A good review of the comparisons between pandas and sql:
http://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html

One can still inspect and manipulate the sql database using pythonic tables; which I do employ throughout the code of the flask. It is a good way to create visualizations and sub/merged tables without using explicit sql syntax which is stricter than pythonic ways. Packages such as wordcloud, scipy, scikit-learn, matplotlib, seaborn are all available to the DataFrame for visual manipulation. If this database had more numeric data, one can format the DataFrames in such a way to utilize the statistical packages and scikit-learn for data mining and even machine learning which I have done in my past work.



In the creation of the database, itself, the most time-consuming and thus most-likely candidate for improvement would be the parsing of the text files to add to the database. The original files supplied for the database followed no patterns and thus proved difficult to parse correctly.
The code to do so, that is create a DataFrame with basic columns which are further parsed and formatted before being entered into the database and the files for the Abstracts are located in both the local and AWS instances:

- https://github.com/noveroa/DataBases/DataBaseParsing/
  https://github.com/noveroa/DataBaseAWS/Abstracts_Gorton)

To initialize:

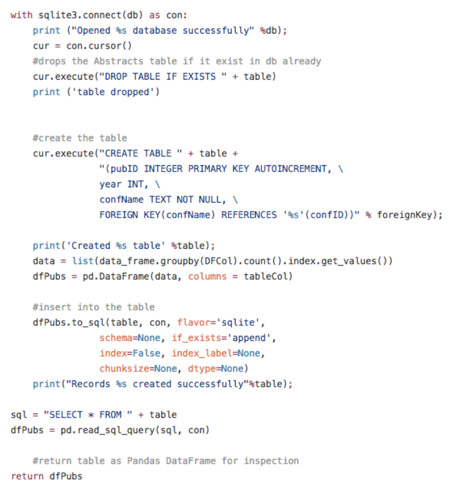
- python intialiazeDBstore.py /<AbstractsFolder>/*.txt
- You will have a .hdf5 file from which to import to create the sql database.

The code parsefiles.py is a script that begins the manipulation of the text files into a single DataFrame saved in an hdf5 file. As the database expands, one may consider writing to a csv file instead. There is much room for refactoring and improvement.

The next step was to create a sqlite database from this python DataFrame. Again the Pandas documentation offers a great path to begin to connect the two. It is nice to have the chance to create a sql database while being comfortable with python and not needing to concentrate on the correct sql syntax, thus allowing for quicker inspection and manipulation (scripts/sqlcommands.py). One can recreate the database running python load_createSqlDB.py found in scripts/.
http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_sql.html

Other than the great Stack Overflow, another great site is from Sebastian Raschka is the author of the bestselling book "Python Machine Learning," whose work I have looked at before.
http://sebastianraschka.com/Articles/2014_sqlite_in_python_tutorial.html

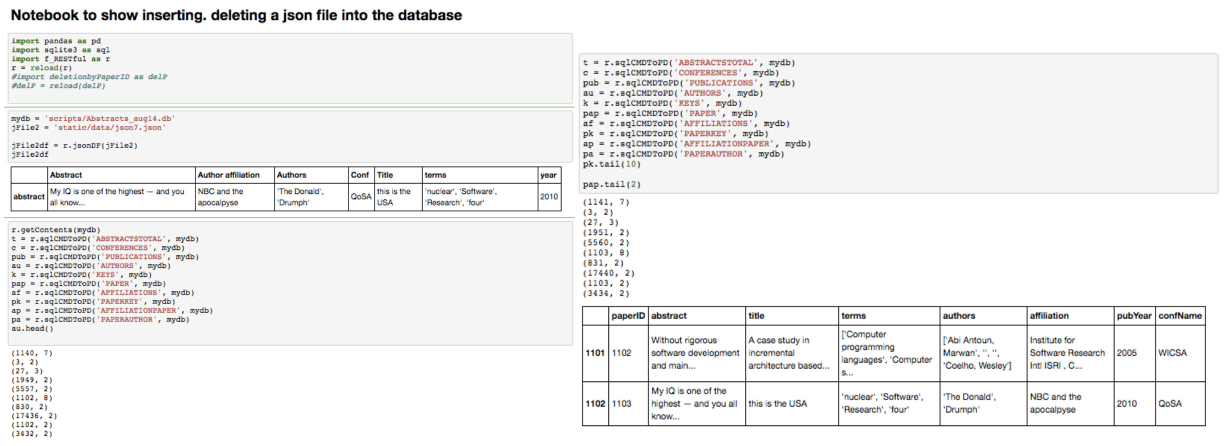
Once you have the columns of the DataFrame as you want them, you can simply read them into Sql. Sql does support foreign key and primary key with auto-increments. For example:



```
with sqlite3.connect(db) as con:
    print ("Opened %s database successfully" %db);
    cur = con.cursor()
    #drops the Abstracts table if it exist in db already
    cur.execute("DROP TABLE IF EXISTS " + table)
    print ('table dropped')


    #create the table
    cur.execute("CREATE TABLE " + table +
               "(pubID INTEGER PRIMARY KEY AUTOINCREMENT, \
               year INT, \
               confName TEXT NOT NULL, \
               FOREIGN KEY(confName) REFERENCES '%s'(confID))" % foreignKey);

    print('Created %s table' %table);
    data = list(data_frame.groupby(DFCol).count().index.get_values())
    dfPubs = pd.DataFrame(data, columns = tableCol)

    #insert into the table
    dfPubs.to_sql(table, con, flavor='sqlite',
               schema=None, if_exists='append',
               index=False, index_label=None,
               chunksize=None, dtype=None)
    print("Records %s created successfully"%table);

sql = "SELECT * FROM " + table
dfPubs = pd.read_sql_query(sql, con)

    #return table as Pandas DataFrame for inspection
return dfPubs
```

It must be noted, now, that the current iteration of the database DOES NOT utilize foreign keys appropriately. This was only discovered in the final week of the project. For sqlite, each time the connection is made to the database the foreign keys must be explicitly turned on. There was a start to this (explicitly calling con.execute("PRAGMA foreign_keys = ON") with each connection; however, key errors and bugs are persistent and thus being not necessarily of high priority (One can do a cascade delete the long way – which is employed currently (f_deletionbyPaperID.py). BUT SHOULD be used for the next iteration of the design.

For each individual table of the relational database the tables are created by first parsing the given columns and reformatting before using. Again this was the hold up of the project. Due to the nature of the given text files, Unicode must be respected which is not a default for python – regex can be used in python and is utilized thusly. This is where many bugs still persist in the database. Much code is spent between Unicode and strings and attempting to resolve such issues. These, however, were inherent from the method chosen to parse the original abstract files – admittedly the file parsing should be reinvestigated, json files may be the key.



Notebook to show inserting. deleting a json file into the database

```
import pandas as pd
import sqlite3 as sql
import f_RESTful as r
r = reload(r)
#import deletionbyPaperID as delP
#delP = reload(delP)
```

```
mydb = 'scripts/Abstracts_aug14.db'
jFile2 = 'static/data/json7.json'

jFile2df = r.jsonDF(jFile2)
jFile2df
```

	Abstract	Author affiliation	Authors	Conf	Title	terms	year
abstract	My IQ is one of the highest — and you all know...	NBC and the apocalpyse	'The Donald', 'Drumph'	QoSA	this is the USA	'nuclear', 'Software', 'Research', 'four'	2010

```
r.getContents(mydb)
t = r.sqlCMDToPD('ABSTRACTSTOTAL', mydb)
c = r.sqlCMDToPD('CONFERENCES', mydb)
pub = r.sqlCMDToPD('PUBLICATIONS', mydb)
au = r.sqlCMDToPD('AUTHORS', mydb)
k = r.sqlCMDToPD('KEYS', mydb)
pap = r.sqlCMDToPD('PAPER', mydb)
af = r.sqlCMDToPD('AFFILIATIONS', mydb)
pk = r.sqlCMDToPD('PAPERKEY', mydb)
ap = r.sqlCMDToPD('AFFILIATIONPAPER', mydb)
pa = r.sqlCMDToPD('PAPERAUTHOR', mydb)
au.head()
```

```
(1140, 7)
(3, 2)
(27, 3)
(1949, 2)
(5557, 2)
(1102, 8)
(830, 2)
(17436, 2)
(1102, 2)
(3432, 2)
```

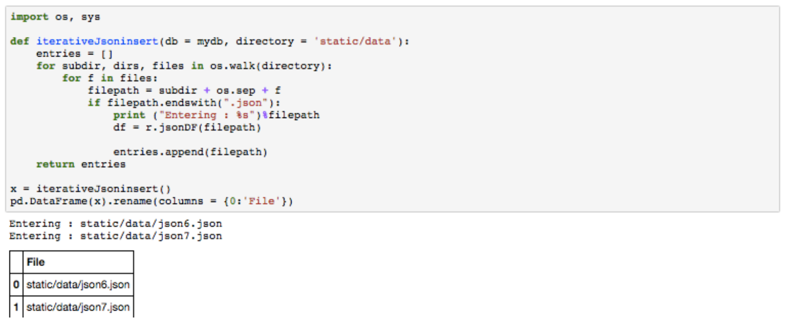
```
t = r.sqlCMDToPD('ABSTRACTSTOTAL', mydb)
c = r.sqlCMDToPD('CONFERENCES', mydb)
pub = r.sqlCMDToPD('PUBLICATIONS', mydb)
au = r.sqlCMDToPD('AUTHORS', mydb)
k = r.sqlCMDToPD('KEYS', mydb)
pap = r.sqlCMDToPD('PAPER', mydb)
af = r.sqlCMDToPD('AFFILIATIONS', mydb)
pk = r.sqlCMDToPD('PAPERKEY', mydb)
ap = r.sqlCMDToPD('AFFILIATIONPAPER', mydb)
pa = r.sqlCMDToPD('PAPERAUTHOR', mydb)
pk.tail(10)

pap.tail(2)
```

```
(1141, 7)
(3, 2)
(27, 3)
(1951, 2)
(5560, 2)
(1103, 8)
(831, 2)
(17440, 2)
(1103, 2)
(3434, 2)
```

	paperID	abstract	title	terms	authors	affiliation	pubYear	confName
1101	1102	Without rigorous software development and main...	A case study in incremental architecture based...	['Computer programming languages', 'Computer s...	['Abi Antoun, Marwan', '', '', 'Coelho, Wesley']	Institute for Software Research Intl ISRI , C...	2005	WICSA
1102	1103	My IQ is one of the highest — and you all know...	this is the USA	'nuclear', 'Software', 'Research', 'four'	'The Donald', 'Drumph'	NBC and the apocalpyse	2010	QoSA

Future Improvements:

The script for creation of the sql database has not been refactored optimally. The abstract text files and the difficulty to parse them correctly was hard to overcome. Hardcoding was removed in most instances, but can be improved. The chance to change the input (ie using JSON files) seems to render this point moot. This is the best way to enhance the speed, cleanness, and ease of the database. With the ability to now insert individual and multiple pseudo json abstract files directly into the database (ie using tags etc) from the running website, the expansion can be done much quicker and cleaner. With the successful implementation of the sqlite foreign keys, deletion will also be improved. Currently, deletion is done via paperID and a long cascade from that (deletionbyPaperID.py).



```
import os, sys

def iterativeJsoninsert(db = mydb, directory = 'static/data'):
    entries = []
    for subdir, dirs, files in os.walk(directory):
        for f in files:
            filepath = subdir + os.sep + f
            if filepath.endswith(".json"):
                print ("Entering : %s")%filepath
                df = r.jsonDF(filepath)

                entries.append(filepath)
    return entries

x = iterativeJsoninsert()
pd.DataFrame(x).rename(columns = {0:'File'})
```

```
Entering : static/data/json6.json
Entering : static/data/json7.json
```

	File
0	static/data/json6.json
1	static/data/json7.json

Also, it should be explored whether REST api and sql commands are faster (both with regards to time and space) versus the creation and manipulation of Pandas DataFrames. Due to the limited size of the original database, this was not the focus of the project and thus not optimized for it.

*****Regarding the AWS instance in particular for the Relational Database:***

Numerous packages are required to be installed. Due to the environment of Ubuntu/Unix pythonic packages can be difficult to import as is creating virtual environments to ensure proper packages and versions. Currently, the AWS deployment does not create and populate its own Relational database due to limits/instability of the Unix PyTables module. Thus it is currently created in the local and uploaded – ensuring to change permissions to the sudo Ubuntu user and allowing Read and Write. Perhaps, one should explore the use of AWS RDS. Another alternative would to utilize Docker instances to maintain environments. I did begin to explore this as well.

The Flask Web Application
Queries and Visualizations

Why did I use Flask? Why not! It is a great tool and often used by companies to host lovely websites. Plus, you can launch a relational database and use python! It is compatible with many languages, has a great community, and can interface with many different packages and languages – including python and using json tools (jsonify!). And it is fun! "*Flask is a microframework for Python based on Werkzeug, Jinja 2…*"
http://flask.pocoo.org/docs/0.11/

Flask
web development, one drop at a time
overview // docs // community // snippets // extensions

Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions. And before you ask: It's BSD licensed!

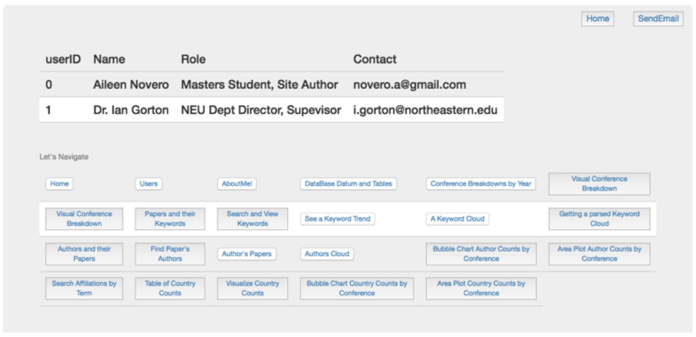
Flask is Fun

Latest Version: 0.11

```
from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello World!"

if __name__ == "__main__":
    app.run()
```

Following the great tutorial available in the Flask documentations, utilizing Flask, jQuery, bootstrap, CSS, and the Python packages matplotlib and Seaborn the web app of the relational database was launched. The following queries and visualizations are currently available among others as they are constantly being added (I can't leave it be for more than a day if even!) – just send me an email! It is not the prettiest site, but please just explore!

Most of the tables support automatic searching and filtering, reordering the columns and rows interactively as well.


Home
SendEmail
Let's Navigate
Home
Users
AboutMe!
DataBase Datum and Tables
Conference Breakdowns by Year
Visual Conference Breakdown
Visual Conference Breakdown
Papers and their Keywords
Search and View Keywords
See a Keyword Trend
A Keyword Cloud
Getting a parsed Keyword Cloud
Authors and their Papers
Find Paper's Authors
Author's Papers
Authors Cloud
Bubble Chart Author Counts by Conference
Area Plot Author Counts by Conference
Search Affiliations by Term
Table of Country Counts
Visualize Country Counts
Bubble Chart Country Counts by Conference
Area Plot Country Counts by Conference

userID	Name	Role	Contact
0	Aileen Novero	Masters Student, Site Author	novero.a@gmail.com
1	Dr. Ian Gorton	NEU Dept Director, Supevisor	i.gorton@northeastern.edu

Extras:
- About the site author: Aileen Novero
    - Links to contact information, GitHub account, Linked In Profile, and Clickable email link
- User page, contact information of site author and academic supervisor for the project
- Welcome page
- Index/Home Page: with links to various tables and visualizations.

Authors:
- Authors and their Papers
    - View all authored papers
    - Author Name, Paper Title, Conference, Year Published, Author's Total Publication Count,
- From Conference Links:

        - View published authors given a conference and year (searchable)
        - Author Name, Paper ID number, Paper Title, Author's Count in total papers published per that given year, Link to view the paper, Link to delete the Paper
- Author's Papers
    - Search for and view author's metadata of queried author's name
    - Author Name, Paper ID, Paper Tile, Conference, Year Published, Delete the Author *(not currently working)*
- Find a Paper's Authors
    - Find the Authors based on querying PaperID.
    - Paper ID number, Author Name, Paper Title, Conference, Year, Author's Count in total papers published per that given year, Link to view the paper, Link to delete the Paper
- Visualize the top overall 20 Authors broken down by count per conference per year:
    - Area Plot Author Counts by Conference
    - Bubble Chart Author Counts by Conference
- Authors Cloud
    - Visualize all Authors as a frequency based WordCloud over all years and conferences

Conferences:
- Search Conferences
    - Search by pull downs, for and view metadata a queried conference and year
    - Paper ID, Paper Titles, Abstract, Link to view the paper in total, Link to Delete the Paper entry from the Database.
- Conference Break Down By Year:
    - View list of each conference and year with links to the published papers, authors, and top keywords
        - View papers published of given conference and year (searchable)
            - Paper ID, Paper Tile, Abstract
        - View authors published of given conference and year (searchable)
        - View top keywords of given conference and year (searchable)
- Visualize Conference Break Down:
    - Visualize each conference broken down by year
    - Table of the publication count, Pie Chart breakdown, and Bar Plot breakdown

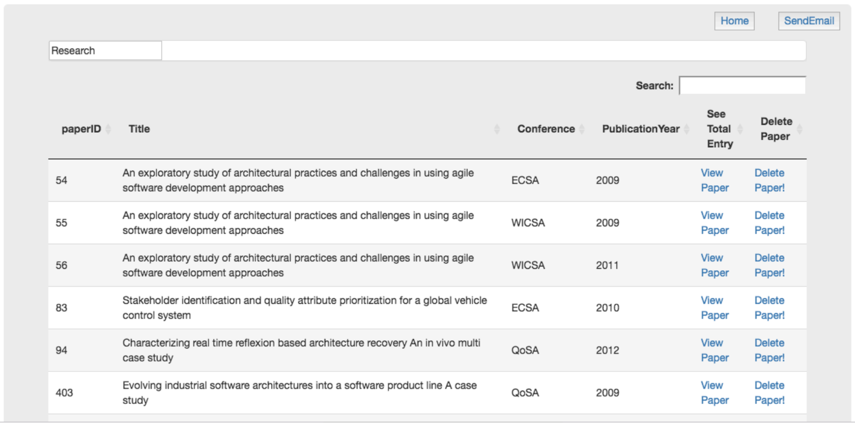
Papers/Affiliations:
- Various Links allow to view metadata of Paper
    - Paper Title, Conference, Year Published, Abstract, Link to Paper's Authors, Link to Delete the Paper
- Search Affiliations by Term
    - View paper metadata from queried affiliation term
    - Paper Id, Affiliations, Link to paper information, Link to Delete the paper
- Table of Country Counts
    - View Country by Count table
    - *Caveat: string search is the way this is implemented.*
- Visualize Country Affiliation by Paper Counts:
    - Bar Chart
    - Area Plot
    - Bubble Chart

Papers/Keywords:
- Papers and their keywords
    - View and search through papers and the list of their keywords
    - PaperID, Keywords list, Link to the Paper Information
- Search for a Keyword
    - Query for a Keyword and View keyword metadata
    - Paper ID, Paper Title, Conference, Year Published, Link to the Paper Total Entry, Delete the Paper
- See a Trend

  - Search for and view metadata trend data queried keyword
  - Table of metadata (paperID, title, conference, and year), Table of Keyword count per conference/year, link to A Heat Map rendering of breakdown
- See Top 20 Trend
  - View table of Top 20 keywords with metadata and link to HeatMap rendering
  - Visualize overall Top 20 keywords broken down by year and conference in HeatMap
- A Keyword Cloud
  - Visualize keywords as a frequency based WordCloud
  - Over all years and conferences
- Getting Parsed Keyword Cloud
  - Cloud Broken down by year
  - Cloud Broken down by conference
    - Can be implemented for authors in future?

JSON/REST Implementations
- JSON files stored in the static/data folder:
- View the JSON file as a pandas dataframe
- Insert the JSON file into the Database
- Delete any abstract via paperID (links in various tables above)
  - CASCADE delete is worked around (currently works for the Key tables and Affiliation primary tables, other than just the composites)
- Delete a table row by element (pk or value; manually employed not cascading)

Tables, Relational Database Returns (all searchable)
- Total Table
  - Abstract, Author affiliation, Authors (nonparsed), Conference Name, Paper Title, Keywords (nonparsed), Publication Year
- Abstracts
  - Paper Title, Year Published, Conference Name, Abstract
- Affiliations
  - Affiliation ID, Affiliation Text
- Authors
  - Author ID, Author Name
- Conferences
  - Conference ID, Conference Name
- Keywords
  - Keyword ID, Keyword
- Papers
  - Paper ID, Paper Title, Publication Year, Conference Name, Authors, Affiliation, Abstract, Keywords
- Publications
  - Publication ID, Conference Name, Publication Year
- Composite Tables
  - AffiliationPaper
    - Paper ID, Affiliation ID
  - PaperKey
    - Paper ID, Keyword ID
  - PaperAuthor
    - Paper ID, Author ID
- Sqlite_sequence
  - A table of the primary tables (non composite) and their counts.



Home SendEmail

Research

Search:

paperID	Title	Conference	PublicationYear	See Total Entry	Delete Paper
54	An exploratory study of architectural practices and challenges in using agile software development approaches	ECSA	2009	View Paper	Delete Paper!
55	An exploratory study of architectural practices and challenges in using agile software development approaches	WICSA	2009	View Paper	Delete Paper!
56	An exploratory study of architectural practices and challenges in using agile software development approaches	WICSA	2011	View Paper	Delete Paper!
83	Stakeholder identification and quality attribute prioritization for a global vehicle control system	ECSA	2010	View Paper	Delete Paper!
94	Characterizing real time reflexion based architecture recovery An in vivo multi case study	QoSA	2012	View Paper	Delete Paper!
403	Evolving industrial software architectures into a software product line A case study	QoSA	2009	View Paper	Delete Paper!

For the future.

One will note the flaskapp is currently run via one python script (welcome_flask3.py and flaskapp.py on the local and AWS instances respectively). Blueprints can be utilized for this purpose, perhaps. But that was not the focus of the project.

Succinctness of the code, refactoring can be improved as well. This was begun and has made for a bit cleaner code, but improvement is always helpful!

Extendibility will come through the Json and REST api most likely. Such scripts can be found in the folder scripts.

Images/Visualizations:
The reason python and DataFrames were originally employed for the database is the ease and breadth of the data visualization packages available.
Due to the way some images are rendered (statically saved versus dynamic), there is collision potential between multiple users. This was handled effectively in the local instance originally, however moving to AWS caused issues and thus the dynamic rendering was lost – especially with the word cloud rendering. This too causes load times to suffer. Too, to ensure, especially on the AWS instance of the correct image, one must reload the page due to caching bug.

File	Reason
welcome_flask3.py	Run the Flask
f_images.py	Image scripts (Bar Chart, HeatMap, Area Plot, Pie Chart, Bubble Chart …)
f_wordcloudmaker.py	WordCloud functions
f_RESTful.py	Json inserts and deletes Querying the database
f_deletionbyPaperID.py	Deleting papers from the database Cascade Delete by Paper ID
f_SqlScripts.py	Pythonic SQL scripts
in Scripts/	
load_createSqlDB.py	Run From Command line to build the Database (linked to .hdf5 file of the First Parsing of the text files)
sqlcommands.py	Parses the initial DataFrame from the .hdf5 file and builds the sqlite relational database
terms.py	Parsing of the terms.
Also in Scripts/ there are a few more sample scripts	

Amazon Web Services
Public Deployment with Ubuntu Server

With Professor Gorton as the instigator, AWS via Ubuntu server, was utilized to publicly deploy the database. To explore in real time, one can point the browser to 54.201.113.183.

The basic approach for the initialization the instance followed:
http://www.datasciencebytes.com/bytes/2015/02/24/running-a-flask-app-on-aws-ec2/

Focus on the mod_wsgi and apache configs – here are the most likely source of errors!

Some key commands to remember:

Command	Reason
ssh -i <abstractsPK.pem> ubuntu@54.201.113.183	SSH into the instance with the proper .pem/.ppk
sudo apachectl restart	If changes are made in the codes while on the AWS instance, to update the live site
While on the remote instance: on remote: ~/flaskapp/ ipython notebook --no-browser --port=8888 *Restart a new terminal:* ssh -L 8000:localhost:8888 -i Desktop/abstractsPK.pem ubuntu@54.201.113.183 *Point your browser to* localhost:8000	Launch a Jupyter notebook as an IDE of choice on the local while SSH'ed into Ubuntu.
/var/log/apache2/error.log	Error log! Oh bugs….
source /home/ubuntu/.bashrc	Updating the bash
~/etc/apache2/sites-enabled/000-default.conf	Location of the mod_wsgi configs.
curl http://169.254.169.254/latest/meta-data/public-ipv4	Get the public ip address
grep nameserver /etc/resolv.conf	Get the private ip address
dpkg --get-selections | grep -v deinstall	Get Ubuntu packages
sudo apt-get install <package>	Install Ubuntu packages

The most important idea to remember, and perhaps only due to my naiveté, for the AWS instance were permission and package issues.

If one is running into an error:
1. Check permissions!
2. Check the path!
3. Caching.

Computer_Software
Computer_Programming
Data_Handling_and_Applications
Buildings_and_Towers
Computer_Applications
Computer_software_selection_and_evaluation
Software_architecture
Software_engineering_techniques
Information_Dissemination
Architecture
Management
Software_system
Computer_Systems_and_Equipment
Digital_Computers_and_Systems
Information_Services
Radio_and_Television
Mathematics
Telecommunication__Radar
Structural_Design
Software_design
Quality_attributes
Engineering_Graphics
Production_Engineering
Industrial_Engineering_and_Management
Design_decision
Automata_Theory
Engineering_Research
Model
Quality_Assurance_and_Control
Software_management
Architectural_design
Reference_architecture
Architecture_description_language
Computer_Peripheral_Equipment
Data_Processing_and_Image_Processing
Software_architecture_design
Systems_Science
International_conferences
Service_oriented_architecture_
Domain_specific_languages
Object_oriented_programming
Architectural_models
Component
Software_performance_evaluation
Engineering_Profession
Architecture_decision
software_quality
adaptive_system
Research
Probability_Theory
Education
Architecture_description
Information_Science
Software_product
Software_intensive_system
Optical_Communication
Formal_specification
Set_Theory
Middleware
Software_Product_Line
Information_technology
Model_transformation
Combinatorial_Mathematics
Meta_model
Mathematical_models
Quality_control
Logic_Elements
Product_line_engineering
Embedded_system
SOA
Design_Pattern
Computer_aided_design
Empirical_studies
Architecture_evaluation
Programming_Theory
Architecture_framework
Product_line_architecture
Software_engineering
Quality_requirements
Architectural_pattern
Quality_of_service
Equipment_and_Techniques
Switching_Theory
Information_management
Performance_analysis
Component_based
Domain_specific
Service_Oriented
based_system
Specification
Survey
Decision_making
Variability
Integration
Software_architect
Requirements_engineering
Software_component
Architectural_description
Ecology_and_Ecosystems
System_architecture
Software_development
Impact_of_Technology_on_Society
Information_systems
Information_Retrieval_and_Use
Software_reliability
Computer_Programming_Languages
Information_Sources_and_Analysis
Architectural_style
Computer_Theory
Computer_software_reusability
Codes_and_Standards
Tool_support
Distributed_computer_systems
Driven_Engineering
Robotics
Industrial_case_study
Network_architecture
Knowledge_management
Performance_prediction
Distributed_system
testing
Information_Theory_and_Signal_Processing
Unified_Modeling_Language
oriented_programming
Knowledge_based_systems
Research_challenges
Product_development
General
User_interfaces
Artificial_Intelligence
Architectural_element
Development_process
Computer_software_maintenance
Includes_Formal_Logic
Project_management
Control_Systems
Software_metrics
Architectural_knowledge
Quality_of_softwares
Ontology
Visualization
Model_checking
Architectural_levels
Cloud_computing
Context
Chemistry
Diagnostic
Industrial_Engineering
Production_Planning_and_Control__Manufacturing
Software_Evolution
Telephone_Systems_and_Related_Technologies__Line_Communications
Cost_and_Value_Engineering__Industrial_Economics
Database_Systems
Maintenance
Architectural_view
Industrial_Economics
Optimization_Techniques
Distributed_systems
Quality_assurance
Engineering_education
Reusability
specification_language
Architecture_design
Architectural_decision
Runtimes
Calculus
Design
Automatic_Control_Principles_and_Applications
debugging_and_evaluating_systems
Embedded_software
Legacy_systems
Interoperability