DB Project ‘17

Group 17: 3:30pm
Erika & Vincent

App Name: TravelBud
Source Code: https://github.com/erikahairston/dbproject2017

Introduction/Motivation: Currently, when people search for Airbnb’s, they only have other users’ ratings and reviews of the home to go off of. If they are not familiar with the area, then it’s hard to gauge how enjoyable the overall experience of staying at the place will be. We seek to provide an added dimension to Airbnb search/browsing by incorporating Yelp data. In deciding where to stay, it might be useful to know whether there are restaurants, businesses, and activities that you like nearby.

Given a desired city and set of other parameters (price, # people, rating) , we query the Airbnb database for listings that those requirements. We then poll the user for his/her interests (up to two each of cuisines, activities, and entertainments). We then aggregate the Yelp businesses within 3 miles of each Airbnb listing to provide a score (currently this score simply consists of the total number of stars that those Yelp businesses have in total, so quantity can overpower quality, but this score could be enhanced in the future). Finally, we return the five Airbnb’s with the top scores, and for each of those we provide the top-rated nearby Yelp business in each category.

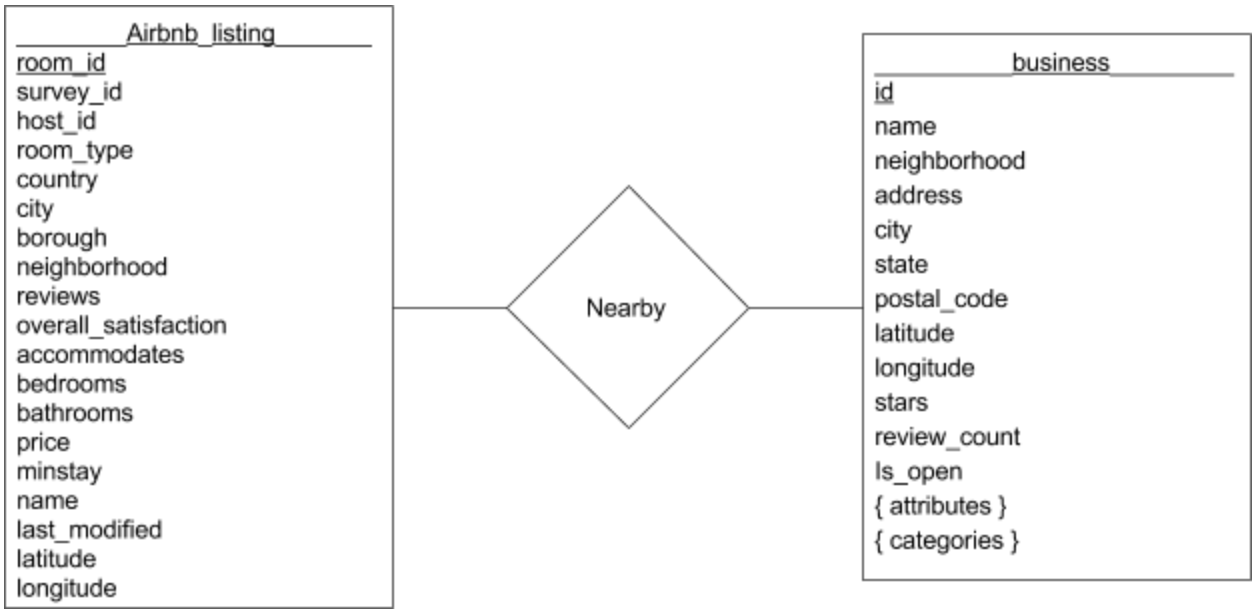
As part of our efforts to provide more information about Airbnb’s and Yelp businesses, we’ve also added on a number of fun facts about the listings, which can be filtered by location.

Challenges:

- Large Tables
    - In having such large data sets, we were pressed to write queries and operations in an optimized way. Some methods that we used were sorting data and using binary search to find the relevant range to work on, as well as using hash tables to represent boolean data instead of having them as a parameter in the query set.
- Open-source dataset inconsistency and incompleteness
    - Some of the Airbnb data formatting was inconsistent. In some cases we had to manually format data in the CSV files.
    - The Yelp dataset only provides data from a very limited number of metropolitan areas (no huge cities, either). The only ones that overlapped with our Airbnb data are Las Vegas and Pittsburgh.

- Django Framework
    - Neither of us were familiar with Django heading into the project. We encountered several major obstacles, some of which stemmed from our lack of familiarity, and others that seem to be deficiencies that are inherent to Django.
        - We had issues importing our Airbnb and Yelp data into Django. When importing schema with foreign keys, we ran into some bugs and ended up deciding just to leave out the foreign key requirements in our practical schema and rather just to treat them as such when working with them.
        - Despite looking extensively into it online, we weren't able to find a reasonable way to get the cartesian product/cross join of multiple tables using Django. The only sensible of cross-joining we found only applied to relations where one's id was the foreign key's of the other. We were also unable to get the Cartesian product of two unrelated keys (Airbnb's and businesses), so we just queried them and operated on them with for loops.

Design:



Our database essentially only requires as entities Airbnb listings and Yelp businesses. In our ER diagram, we include a relation "nearby" between them because locality is the only sort of way they interact within our app. However, even this isn't really the case because within our implementation we get this relationship by effectively cross joining the two tables, rather than having another relation to map listings to nearby businesses, or vice versa. So, arguably our ER diagram actually is just our two entities with no relationship connecting them.

Our database design implementation decisions were greatly influenced by the fact that we had already committed to using Django. As described in challenges, we ran into a number of

limitations in using Django, such as the inability to join relations that are not connected by a foreign key, and to join more than two relations that are connected by a foreign key. We also had troubles importing tables with foreign keys in them. Thus, we went with an implementation that was not necessarily the best database design.

If viewed with certain conditions, our database design is in normal form. This hinges on whether we consider the location-related attributes as having dependencies. There are arguably dependencies for Airbnb listings neighborhood → borough, borough → city, city → country, etc. There are equivalent potential dependencies in our businesses relation. If this is the case, then we would need to move the location-related attributes into separate tables. On the other hand, one can justify that there is no dependency between the location attributes. Two neighborhoods of the same name can exist in different cities, i.e. Chinatown. Cities with the same name can be found in different countries. A postal code region might cross state lines. And so on. So if this is the case then the location-related functional dependencies wouldn't hold, the only non-trivial functional dependencies are of the type room_id/id → [subset of R], and so all of them would satisfy the 3NF/BCNF condition of having the left side be a superkey. We do acknowledge that a location dependency that is more certain to actually exist is that between state and country, since states have unique codes that won't be shared even if a same-named state exists in another country. We had planned to do have a separate country table to make this into normal form, but we had assigned it to one of our team members who dropped, and didn't end up revisiting it.

Also worth discussing is our handling of the multi-valued attributes Categories and Attributes in the Yelp business relation. In order to make our design normal form, we created separate category and attribute relations to map businesses with categories and attributes. Though we left out the foreign key constraint in our models because of our struggles importing our data, we operated on them as if the foreign key constraint were there.

Installation:

1. cd into djangoProj/mysite
2. unzip db.sqlite3.zip, export to . [djangoProj/mysite]
3. make sure you have pip and django installed
    a. pip: https://pip.pypa.io/en/latest/installing/#installing-with-get-pip-py
    b. django: https://docs.djangoproject.com/en/2.0/topics/install/
4. Install geopy
    a. run: python -m pip install geopy
5. run: python manage.py runserver
6. in browser, go to http://127.0.0.1:8000/polls/

Notes:

1. Because of the incompleteness of the Yelp data set, the only cities for which the Airbnb listing search will return meaningful results are Las Vegas and Pittsburgh. However, feel free to use the secondary search section to query fun facts about the Airbnb's in each city.
2. At this point, we have limited interests selection to two cuisines, activities, and entertainments, each.
3. We have extraneous Django models (relations) because we based them off the original data sets but didn't end up needing them. Then, we chose not to delete them so as not to invoke further migrations. The only relevant models are Airbnb_listing, business, attribute, and category.
    a. In the future / if we had more time, we would have used our attribute table to filter through businesses that are best-suited to the user. For example, if the travelers will have a car, we will only consider or value more highly businesses that have parking. Or we can filter through restaurants that only have a type of ambience that the user desires.

What Team Member worked on:

Erika	**Vincent**
Setup the Django framework using the polls tutorial	Setup models for Yelp data in order to migrate into Django. Also manipulated Yelp data and converted it into .csv for importing.
Used CSS templates to work on the entire front end and add javascript/jquery functionality	Built out and formatted the third page (/polls/yours/done.html)
Designed and built the first two pages (/polls.html) and (/polls/yours/) and set up message passing between pages	Wrote report
Worked on the fun fact queries: - highest and lowest priced airbnbs - avg. price airbnb - most commonly used word in listing names	Implemented all of the Airbnb (enhanced by Yelp data) search functionality. Iterated over the process multiple times to optimize operations in order to reduce runtime.