Friend Match

Project Documentation & Overview

# Friend Match – Overview

Database derived from [OK Cupid](https://www.kaggle.com/andrewmvd/okcupid-profiles?select=okcupid_profiles.csv). Data cleaned and Normalized tables for friend matching application.

## ER Diagram

The following is an ER diagram of the various entitles in our Database:

Diagram

Description automatically generated

## Person

As you can see a person has a particular profile which lays out their demographic, preferences, and traits. The person also takes (or has taken) a survey which asks questions specific to their likes and dislikes.

The survey taken will be used to identify profiles and their respective peoples for displaying the possible matches.

## Profile

Since many people have similar profiles we seperated the profile attritubes. This reduces storage on the system as we don’t need to duplicate the similar profile attributes. The profile can be broken down more to separate attributes for demographics, traits and preferences.

This breakdown into three separate tables helps grouping of common values and this also saves space. It improves memory utilization to as the elements in each row are integers to the respective attributes.

## Survey

Before going in to the details of this table let me begin by addressing the obvious question, why do we need a separate table since it is identical to the Profile table ? At this time there is no need as it is true that this is the same set up as profile is today. However, we are assuming that the questions asked in the survey are infact the same as what the profile is. By keeping it a separate table we can add items to the survey table such as strong dislikes of certain traits, or add new questions/attributes in the future that may not be relevant to profile. Since each person has an ID of the survey, in the future we can simply change the survey table without the need to touch the person table.

Why is this useful you may ask ? Simply due to the number of persons involved, we can have 50,000 or more entires in person table and doing updates there would require too much resources. Plus the questions in Survey may not be relevant to profile, example could be a value for sponsored survey to indicate that this is a paid for survey.

The survey table for npo matches the same three groupings as profile does, and for now the idea is to find persons whose profileID matches the surveyID as possilble matches.

## Demographics, Traits, and Preferences

The tables themselves are self-explanatory and were split up with the same reasoning as before (normalization). However, we can apply further normalization as we did with the languages table in the traits table. We could normalize demographics to have religion be a foreign key to a new entity called religion with a religionID as a INT. This will allow for easier expansion and reduce further the storage requirements associated with demographics.

During the processing of the raw data we reduced the choices for religion down to agnostic, atheism, Buddhism, Catholicism, Christianity, Hinduism, Islam, Judaism, and other. Technically there was no need to do this since we could have converted the various values given to an INT.

### Future enhancements for Demographics

Diagram, timeline

Description automatically generated

If we had more time we would have considered the following ER to help improve performance. Looking at the diagram you may wonder why we would have a table such as ethnicity, or oritentation split out, especially since it only have 1 element. Doing this is useful because the values we assign are highly dependant on what society uses. Similar can be said of location, for example if a city changes name from Bombay to Mumbai we only need to change 1 row in the Location table, as opposed to the current set up where we would need to update all records in demographics that have Bombay to Mumbai. A similar reasoning would be for ethnicity and orientation can be made.

### Future enhancements for Traits

Diagram

Description automatically generated

The current Traits table has the language attributes seperated but we can enhance this further for all the reasons already discussed. Regarding Height we would move the actual height value to the Person table because this will have lots of values but typically we want to find people based on set levels such as short, average, taller etc. Soring the range instead of the height in the traits table will reduce the number of records by a great deal.

In the above ER diagram we use the Height Range instead of the actual value. This will reduce the number of records for hieghts and will improve the searching and validating inputs.

### Future enhancements for Preferences

Timeline

Description automatically generated

Further improvements can be made by seperating the tables as described above. This will allow for easy expansion of values later. For example we may want to allow new PetsID values to include reptiles. Doing these seperations will allow the input screens to have a drop down for the different values someone can select from.

### Future enhancements for Persons

Currently the number of persons in the data file is around 60,000 and that for now is manageable by having a single Persons table. However, as the number of People grow we would need to make a hierarchical approach to the Persons table. One such approach is to have multiple the Persons Tables with each one representing a certain number of users. So if we have a million users we could have a Persons table for the first 100K, then another one for 101-200K users, and so on. To know which table to refer to we would look in a new Person Table Index which table is the one to use based on the userID provided. The diagram below helps explain this:

Diagram

Description automatically generated

This would allow us to have the Person100 table in a different location to Person900. If the concern is that we want to make it more distributed than the above since the above is more sequential, meaning userIDs are sequentially assigned, we could base the look up to a table on the last digit of the UserID (or the last three). This would mean the Person tables would start to get populated in an even distribution.

# Processing Raw Data and Populating Tables

## Files Delivered:

|  |  |
| --- | --- |
| File | Description |
|  | |
| Python Files | |
| db\_operations.py | This contains operations used to execute SQL commands. Its main purpose is to make the code easier to read. |
| helper.py | This contains useful methods that are common across all projects. |
| main\_final.py | This file contains the main logic of processing the raw data, dropping and creating tables. |
| populator.py | This file contains useful methods for populating the tables and lists used. |
| process\_initial\_data.py | This file will take the raw file (rawdata.csv) and generates the temporary files:   * staging.csv * traits.csv * demographics.csv * profile.csv * survey.csv * person.csv   These csv’s are used to initially populate the tables. The script will ask how many records from the raw data do you want to use. This helps reduce the 60,000 records in raw down to a smaller number you request to allow for quicker development. |
| process\_raw\_data.py | This file is obsolete and should be removed from GitHub prior to final submission. |
| testclient.py | This is a simple client to test the SQL commands and databases via a command line. It is used only for testing purpose. |
|  | |
| CSV Files | |
| rawdata.csv | This is the raw data that we have been given. This contains approx. 60,000 records in a csv file that needs to be process. |
| newdata1.csv | When process\_initial\_data.py runs it asks you how many records you want to process from raw data. This file contains just those records. It is a temporary file used as an in between step. It is outputted to allow for verification that the correct number of records is being used. |
| staging.csv | Generated file from process\_initial\_data.py containing attributes required to update the staging table |

## Notes about the python files:

I utilized pandas library to help filter the input data as this was way more efficient than my original design using dict, list and arrays. Using pandas enabled me to use built in functions to do quick sorts and replace commands. The tricky part was to replace values in a column that did not match a string, since pandas does not do that by default. This required me to do this in a two-step process as can be seen in the code where the use of a unique keyword “JEEVAN” was used and then replaced (see method fixUpReligion() in process\_initial\_data.py as an example).

Since I had to contend with connecting to different instances and different databases mainly because of issues explained later, the file has several commented out IP Addresses and database names.

A staging table is created for initial purposes to allow the population of other tables from that bigger table. Ideally this would be dropped after the populate script is run. We have left it up for now so its easier to perform validation in case anyone wants to check however it is not needed after the initial load.

# Issues Faced

As a team we took the approach that it’s better to start with a reasonable normalization first as we each had to learn and apply the concepts learnt in class. The goal was to ensure we have a working version first and then look to see if we can improve based on time.

## Issues with MySQL and Google

During the development when I was populating the tables using inserts the Google connections hung. They would hang for hours meaning I was unable to perform any operations. Not only those new connections to that instance failed. After trying to troubleshoot this and failing I decided to create a new instance and connect to that. This deployment activity took a few hours while Google did its activation cycle. However, the new instance would suffer the same issues quickly once I started to populate that. In some cases, it prevented accessing the DBs till the next day slowing down the coding and testing. Even restarting (which took an hour) did not resolve the issue.

Looking at several websites and utilized their recommendation to find any issues. Many on the messaging boards mentioned it’s likely that I have exceeded transaction limits and I did see on the dashboard my writes / reads had a chart showing 300%. Assuming I had reached the limit trying to add the 60,000 records from our initial sample I reduced the number of records greatly to ensure the database wont hang. By reducing the number of records, I have not faced this issue so suspect there must be some limit imposed by Google for my instance. I have not looked in to this any further due to time constraints.