**College Data Visualizer**

Helen Zhang and Michelle Lin

*CS 3265: Database Management Systems*

April 2020

**I. Introduction**

*Overview*

College Data Visualizer is an analytical web application developed with flask and bootstrap for the front-end and MySQL as the back-end data source. The application is intended to give users the ability to filter through colleges based certain attributes like SAT scores and location.

*Application*

We choose to build a web application because, out of all the choices, it was the most suitable given our project experience and language knowledge. Moreover, decided to build a filtering application with the college scorecard dataset because there aren’t many resources for students to be able to preview aggregate data during the college search besides individually searching up college websites or filtering by only SAT or act. Our hope is that the application will allow students to not only gain a helpful snapshot of each college, but also the ability to filter based on certain attribute with an intuitive interface.

*Dataset*

The application data is derived from a modified version of the most recently updated (March 30, 2020) College Scorecard distributed by the U.S. Department of Education containing information about all registered US Colleges from tuition case, undergraduate enrollment size to institution website links from the years 1995 - 2018. The year 2012 and 2016 were excluded for database insertion reasons to be discussed later in the report. This dataset was chosen because three reasons. One, the raw data is substantial at roughly 250 MB. The dataset consisted of at least a couple hundred attributes allowing us flexibility to pick and choose those that we felt were relevant and would provide a helpful overview of each college. Three, we thought this data would be helpful for current high school students who are either researching colleges or making college decision during these times where most colleges have closed off their campus and visits have been cancelled.

**II. Final Implementation**

*Description of Final Implementation: use cases/platform*

*Final Database Design*

1. Normalization Process

The database US\_Colleges contains exactly 52 attributes. Here are a couple attribute shorthands to better understand the dependencies:

* UNITID: institutional unit id identified by US Department of Education
* INSTNM: name of institution
* DATA\_YEAR: year data refers to
* CITY/STABBR/ZIP: city/state/zip of school
* INSTURL: college website link
* NPCURL: college financial aid website link
* CONTROL: type of institution

For the sake of space and clarity the simplified function dependencies are listed below:

Violations in **BOLD:**

UNITID 🡪 all other attributes

INSTNM 🡪 all other attributes

**UNITD 🡪 INSTNM**

Thus, a separate basic\_info table was created with only UNITID and INSTNM and UNITID is the main key for all other attributes in the table.

CITY, STABBR, ZIP -> UNITID, INSTNM, INSTURL (and etc)

**UNITID -> CITY, STABBR, ZIP**

Thus a separate geographical\_data was created with UNITID as primary key and CITY, STABBR and ZIP as the only other attributes.

UNITID -> INSTURL

**INSTURL, NPCURL, CONTROL -> all other attributes**

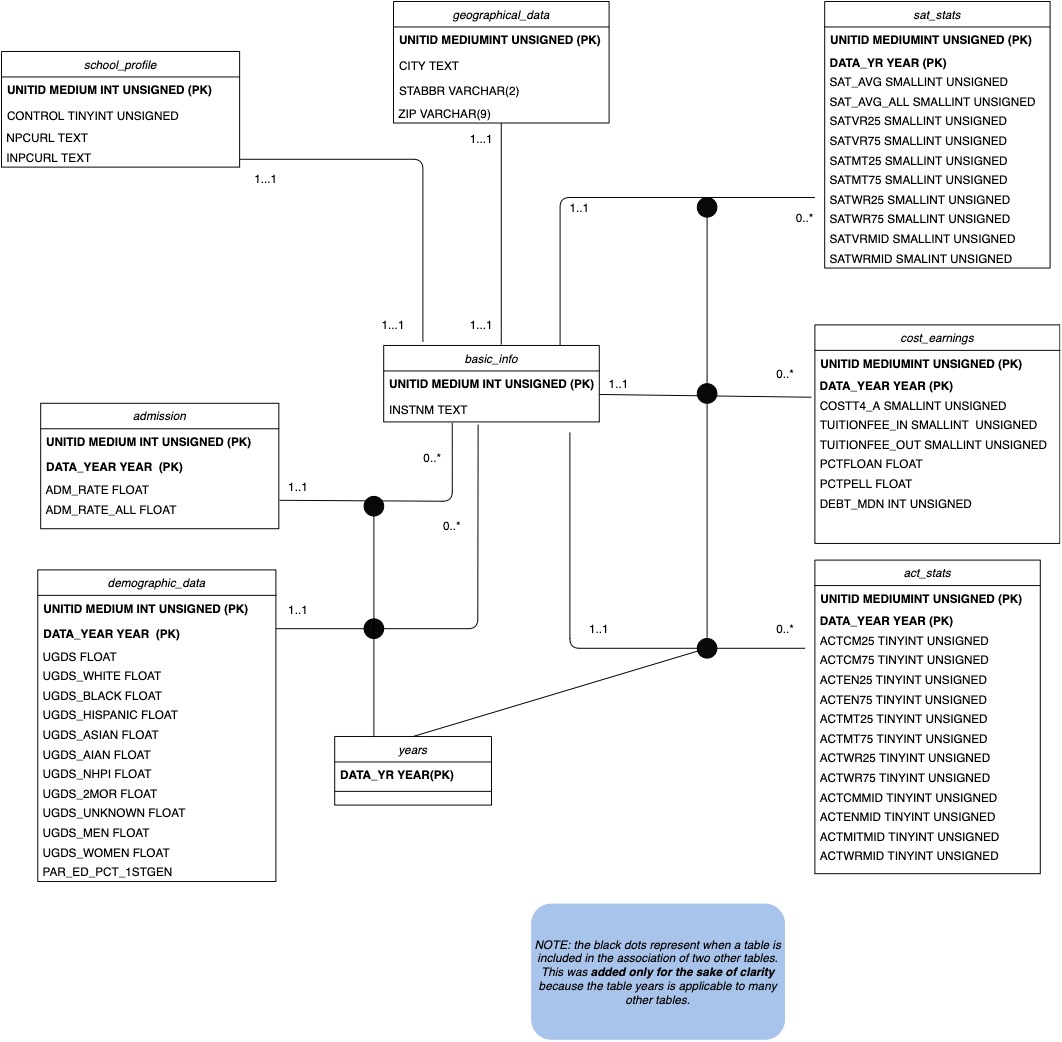
Thus, a separate school\_profile table was created with the four attributes above with UNITID as primary key.

While the attributes in cost\_earnings, sat\_stats, act\_stats and demographic\_data were not directly found to create violations, these tables were created to ease querying and aggregation of relevant data for future data insertions.

2. Final Design

The main attribute for each college is UNITID which is the unit id for each institution. The core table in the database is basic\_info which maps UNITID to the (INSTNM) institution name. Only 2018 data was inserted into this table to avoid discrepancies between INSTNM and UNITID in previous years. This is the case for school\_profile as well as geographical\_data. The other tables sat\_stats, act\_stats, demographic\_data and admission describes data for insitutions for all years listed in the years table. Finding information in these tables requires UNITD and year as there are multiple data points. Allocating a separate table for year is meant to allow for the insertion of multiple years and future additional data.

3. UML with cardinality

**

*Description of Data*

The raw data is disturbed and maintained by the U.S. Department of Education. The data set can be found at this link: <https://collegescorecard.ed.gov/data/>. Since the original data contained a significant amount of attributes (up to hundreds), the data files, each describing college data based on year, was cleaned to produce 20 individual files with only application relevant attributes using python pandas. Each file consisted of approximately 7200 rows of information with each row concerning only one college. The data used for testing consisted only of the 2018 data which had 7243 rows or colleges. This data was used to determine the insertion process from education\_mega (mega table with no normalization) to the normalized tables and determined if there were any dependencies violation between attributes as well as figuring out the semantic of loading the file correctly.

*Summary of Implemented Use Cases*

**III. Illustration of Functionality**

**IV. Summary Discussion**

*Challenges*

On the database side, the biggest challenges were cleaning the data, loading and inserting the data. Since each individual files had hundreds of attributes with thousands of rows, it was difficult to determine relevant attributes as well as produce completely new csv files for each containing the desired attributes. To resolve this, attribute inclusion was decided based on personal experience and the file cleaning was automated using python pandas. The test file 2018 data some anomalies such as non UT-8 encodings and duplicate keys that made it difficult to load the file into the education\_mega table. This was resolved by deleting some rows of data that caused error messages. The insertion process was relatively smooth for just the 2018 data. However, inserting into multiple tables for 20 files was tedious as the insertion statements were needed to be written for each file and each file needed to be check for duplicates keys. To mitigate, atom was utilized to replace each instance of the year data and replicated the insert statements. However, each insert still needed to be manually tested.

*[include front end challenges]*

*Division of Work*

Michelle was responsible for the database side of the project this included: normalization, database design, cleaning raw data, database analysis, UML diagram and the majority of the report. Helen was responsible for the front-end and back-end including: environment setup, connection between all three interfaces, front-end design, retrieving data from the database at the back-end, error-handling on the front and back end, database analysis/modification, video walkthrough and some of this report. This was a fair and equal split because setting up the database and the front-end both required significant amounts of time. Since Helen was responsible connecting all three interfaces together as well as the video, Michelle created most of the project report. Contributions outside of one’s delegated responsibilities were also made.