Final Project: Crimes and Crashes

By Shrineil Patel and Steven He

Driving under the influence has been a prominent crime across the country for years[1]. It often results in a vehicular crash, causing damage to infrastructure or other drivers. This turns out to be an observable pattern and became the focus of our team project. We focused on the Austin area in our study and used the local police records for crimes and vehicle crashes. Figure 1 describes our approach to modeling instances of drinking and driving. The dataset for Crime and Fatal_Crash refer to the two local police datasets. The primary keys from both tables will be used to identify crimes and fatal crashes that occurred at a specific times. This will map out where crimes and crashes occur in unison in Austin, a typical feature of drunk driving. The three other entities described sections of the crash dataset that could be involved in drunk driving. These tables contain information on helmet use, license status, and seatbelt use. These can all be related to any of their corresponding crashes to assess the status of public safety. This same framework was used in the final project to design our twitter queries. We looked for text in tweets that included the term "DWI" and "drunk driving". The number of these tweets will show the amount the current public awareness on drinking. With this concept, we created a logical model to describe a database.

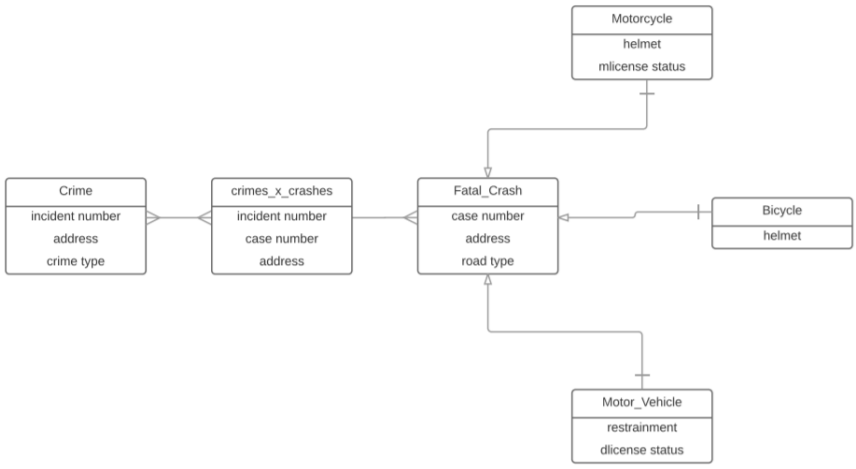

Figure 1: Conceptual model for datasets

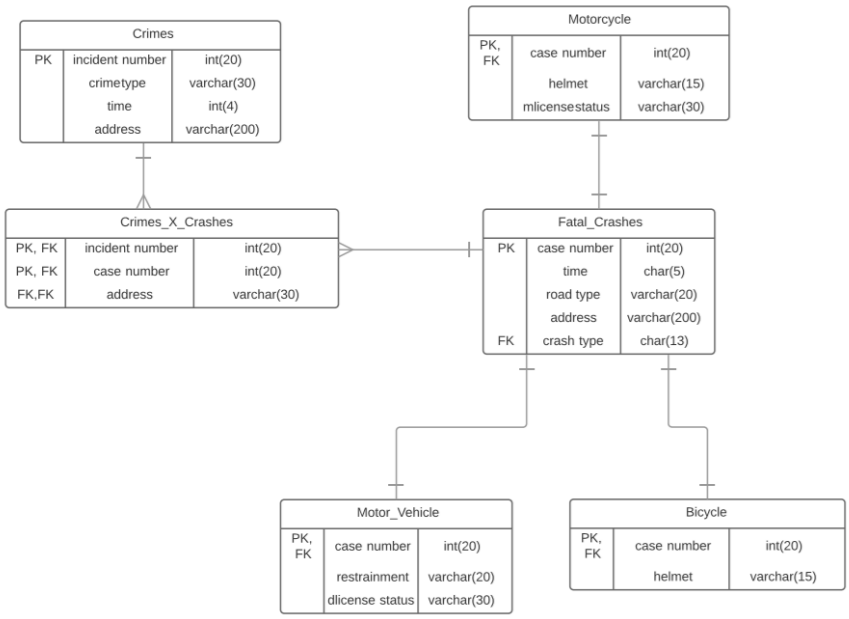

Figure 2: Logical model for datasets

The model described in figure 2 represents the tables to be built in a MySQL server. The crimes table will contain the type of crime committed, time of the crime, the address for the crime, and a unique incident number for each crime that will be used as a primary key. The Fatal_Crashes table will contain the time a crash occurred, the type of road the crash occurred on, the address of the crash, the type of collision, and a unique case number that will be used as a primary key. The Motorcycle, Bicycle, and Motor_Vehicle tables are children of the Fatal_Crashes table that pertain to the vehicles involved in a given crash. The specific attributes they contain are unique characteristics of those crashes like presence of a helmet or driver's license status. The varchar used for address was intentionally set to be large due to varying size in the dataset. The final project added an extra level of depth to the model in the form of public awareness. This involved creating two new tables, one titled Crime_Type, and the other titled Tweets. These tables sought to link the various types of crime listed in the crimes table with public awareness by pulling tweets off of a Twitter API. Once tweets were pulled using crime types as a search term, they were then added to the Tweets table in a JSON file format. The Crime_Type table was linked to the Crimes table by establishing the type_of_crime column in the Crimes table as a foreign key of the Crime_Type

table, and the Tweets table was then linked to the Crime_Type table by establishing the type_of_crime column in the Tweets table as a foreign key of the Crime_Type table as well.

Collecting the data for the MySQL server was done via CSV files retrieved from the Austin Police Department and the Austin Government Data website (https://data.austintexas.gov). Two files were collected; the APD crime report and the 2015 fatal crash spreadsheet. The crime report contained a crime type field, an incident number, an address entry, a date, a time, a longitude, and a latitude. The crash report contained many more fields ranging from day of the week to impairment status. The datasets were combed for the specific data in the logical model via import scripts. For the final project, the twitter API was used to search for public awareness on the prevalence of crimes. Initially, the search in Twitter was done for every crime type located in either the Crimes or Crime_Type table. However, this proved to be fruitless since there were so many crime types located within the database. This resulted in the Twitter API exceeding the rate limit whenever the search or searches were performed, requiring the group to limit the search terms. After much trial and error, it was reluctantly decided that only two types of crimes were to be searched for in Twitter, "CRASH/INJURY", and "RECKLESS DRIVING." Even with only these two terms as the delimiter for the API search, well over fifteen hundred were still returned, resulting in a large and informative database.

The data was loaded by reviewing each line and recording the data found at different sections of the line by using an index. An insert statement was prepared for each line and held the field names and their values. This process was run repeatedly in a for-loop until the end of each data file and stored in a corresponding table in the MySQL database. The Fatal Crash and Crime table were loaded first, followed by the children of Fatal Crash, and finally the junction

table. The primary keys were set for each table as case number or incident number and the primary/foreign key pair was set as a combination of both in the junction table. The address variable was used as the connecting variable for the junction table but time could have also been used. The rollback scripts ran delete statements on every line of a table until the table is empty. A check was run for insert statement validity, commit validity, and import success on each import script. The final project used a duplicate key check to make sure only one record was assigned to each crime type for the import of the Crime_Type table. The import for the project used the same import/cursor system of the labs preceding it.

The set of queries chosen explored various relationships in the data. It used a wide variety of the select statement formats and keywords. The first query pulls data for a specific crime type using user input. The query takes the input from the user and creates a select statement around it. A check is run to make sure no sequel injection text is used in the query. The second query did the same for crash type using an identical method. The third query targets specific data in the motorcycle table. It runs a select count statement for the number of motorcycle crashes without helmets. The fourth query uses a modified version of the fourth query to count the number of crashes that occurred on local streets. The fifth query returns all crashes that match the address of a committed crime and it returns the exact same entries as the junction table. The query uses an inner join between the crimes table and fatal crashes table to produce this result. The sixth query uses an outer left join to pull all the locations without a matching crime. The seventh query uses the “distinct” keyword to list out all the different types of crimes. The eighth query uses an outer left join between the crimes table and the junction table to return the crime files behind the junction table entries. The ninth query runs and inner join between crimes and crashes and returns entries that occurred at the same time. Query ten takes user input to determine a case

number and looks up the matching crime data. Most if not all the queries were set in ascending order by using the "order by" function. The queries for the final project included select all tweets by date created, find screen name and crime type frequencies, select all crimes with type_of_crime fields matching the search fields in the tweets, and finally all tweets with crime type search fields matching those of crimes located within the junction table. These queries did not require any user input, and were performed simply to perform large statistical analysis on the tweets pulled from the Twitter API. Unfortunately, the last query, selecting all tweets with matching type_of_crime fields as the crimes within the junction table, did not yield any results. Analysis of the entries located within the junction table revealed that the crimes located in the junction table indeed did not have any matches with the Twitter queries.

This class has taught me how to logically approach problems. The labs all required a clear line of thought to create a valid schema, a good knowledge of syntax to avoid lengthy debugging, and coordination with a partner. An unexpectedly difficult problem was choosing subject matter for the labs due to the requirements the datasets had to fulfill. After that, most of the work involved testing functions and mastering syntax. Each keyword and function could be used in multiple ways and made the query building very precise. It also was beneficial to make small changes to code and then test instead of coding all at once. The final project was a culmination of the skills learned through the labs and required the use of nearly all of the class concepts. It exposed a more professional side of databases and their uses in reality.

The first lab required my group to find a series of datasets that have a junction table run on them. We ran into problems with the first few databases due to difficulties in finding a partner database. The primary keys overlapped on many of our first attempts at a conceptual model. This would occur due to choosing primary keys like country or vehicle manufacturer. We decided on

a crime database and a vehicle crash database since both had a location variable and different primary keys. Creating the junction table turned out to be the most difficult task in this lab since we had trouble getting the locations to match. Upon completing the assignment, we discovered a flaw in the logic of the junction table. It had used only unique addresses since we had labeled location and case ID as a primary and foreign key respectively. This resulted in a short 12 entry table that contained unique addresses while ignoring repeat addresses. This problem was discovered in lab 3 through the use of select statements, which turned out to be an effective proofreader on import scripts. A slight annoyance arose from the utilization of github because it changed the login information for the MySQL root database when I or my lab partner synced the client. During the final project, the twitter API significantly slowed down progress due to rate limit errors. Each time this error occurred, it would require 15 minutes before we could search the database again. It exacerbated another issue with the API. A search to get data is done blindly with no way of knowing how many files will be pulled. This made formulating queries a very risky proposition due to the inevitable rate limit error.

The value of this class came from building and test our own databases. From picking the datasets to debugging the code, each lab brought a sense of autonomy to my group. The result of our study was inconclusive due to lack of data on one side of the pairing. The dataset for fatal crashes contained far fewer entries than the crimes dataset. This led to a possible exaggeration of crime frequencies at varying locations. If another study were to be conducted in this manner, a few changes would be made to the procedure. The datasets would be checked for similar lengths and cleaned of extraneous text before use. Perhaps a less punishing API would also be used for adding data. All in all, the class and labs have taught my team concepts and practices used in the world of today's data.

References

1. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812231