US Accidents Application

Li Yuan and Matthew Flaherty
DS 5420
Vanderbilt Data Science Institute

Introduction

The application that we have created is an R Shiny app. Shiny allows individuals easy access to create an interactive web app from R. The data that we used is about car accidents in the United States. It contains accidents from 49 states from February 2016 to December 2020. We collected the data from Kaggle and the collector said this data uses two API's that provide streaming traffic incident data. The API's broadcast traffic data captured by a variety of entities. There are 4.2 million accident records in this data with 49 features. We chose an R Shiny application because of its power and our interest in learning something useful. Shiny is powerful because it combines the power of R with the interactivity of a web page. We both had an interest in creating a Shiny app because we can apply this knowledge outside of this project by implementing Shiny apps in our work post-graduation as well as in internships and other projects along the way. We initially began looking at this data set because it fulfilled our requirements for the project. It had more than 30 attributes and was greater than 100 MB. We had other options for data sets; however, those data sets were too large to accomplish what we needed to have completed in the time given and those data sets were not as complete as the one we chose. Ultimately, we chose this data set because we knew we could accomplish solid work in the time given based on our ideas.

With this data set, we wanted to decompose the table using normalization as well as create views, stored procedures, triggers, and other advanced features. Having all of these attributes would allow us to have an easier time implementing the data into our app. In our application, we wanted to find trends in a date range, get a distribution of the temperature, find the number of accidents on a side of the street, find the number of accidents by state, search for observations in the data, update the data, insert into the data, and delete observations in the data.

Final implementation

We used our R Shiny app to find trends in a date range, get a distribution of the temperature, find the number of accidents on a side of the street, find the number of accidents by state, search for observations in the data, update the data, insert into the data, and delete observations in the data. We wanted to find the accidents in a date range because it would be interesting to analyze if any time of year had more accidents than other times of year. The distribution of the temperature would be an interesting

analysis to show the range of temperatures and to also see where the average temperature for the United States was during this time period. We thought it would also be interesting to find the number of accidents per side of the street to see if drivers tended to wreck on one side of the street over the other. Then we wanted to analyze the total number of accidents for each state during this four-year time frame because we could see which state had the most accidents. We thought that our users ought to be able to search for accidents to see if all of the information was correct or to just see an individual accident, so we added a feature where the users could search for accidents. Having searched for accidents, if a value was incorrect, then the user would want to change value, so we added a page where the user could do this. On the other hand, if the user wanted to insert a new accident, we wanted them to have this ability, so we added a page where they could insert an accident. Lastly, we wanted the user to be able to delete an accident for any reason, so there is a page where the user can delete an accident.

We first needed to get our data into first normal form. This required us to make sure that each cell was atomic. Once we had completed this, we still had our mega table as all of the cells were already atomic. Next, we needed to get our table into second normal form. This required us to make sure that every non-prime attribute of the relation was dependent on the whole of every candidate key. The candidate key was the ID of the accidents; therefore, every non-prime attribute was dependent on the candidate key as a whole. We needed to have our table in third normal form, at least, to make sure we fulfilled the normalization requirement. For a table to be in third normal form, there must not be any dependency among non-key attributes. The attribute “country” is dependent on the attribute “state”. Thus, we needed to decompose this into a table where “state” was the candidate key and “country was the other variable in the table.

The final database design contains a location table that contains the states and countries in the data set. The primary key is the states as it determines the country. The other decomposed table is the US accidents which contains information on accidents in the US during 2016 to 2020. The primary key for that table is ID. The mega table is still in the database to hold the data in case we want to change any of the decomposed tables. It allows us to have the data loaded in mySQL.

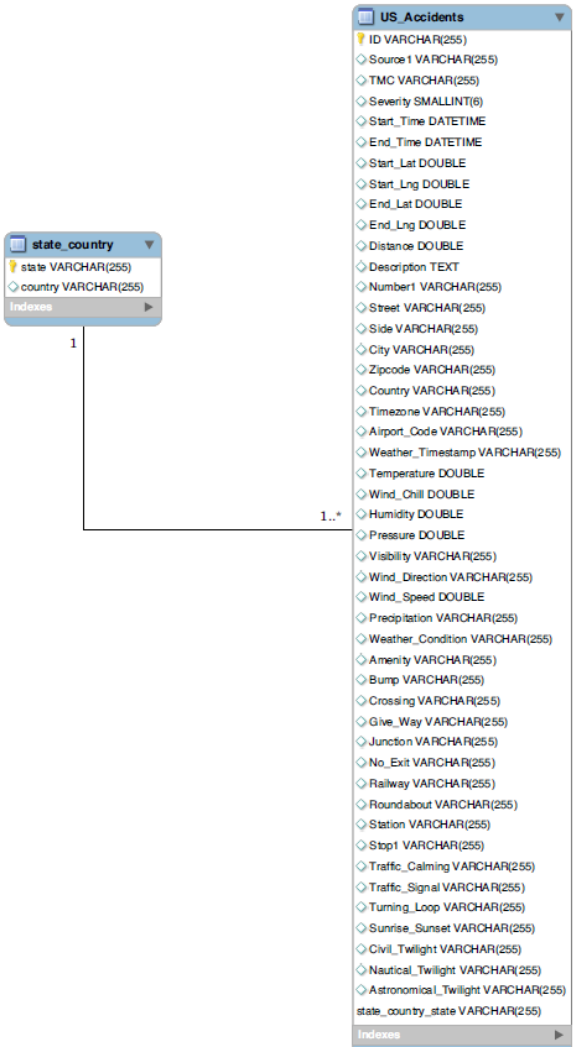
We made four views for our database, “accidents_state”, “common_date”, “common_time”, and “view_table”. “accidents_state” was created to get the total number

of accidents per state which allowed us to use this view in our “Accidents by State” page. “Common_date” was created to get the total number of accidents on a day which allowed us to use this in our line graph in the “Number of accidents by date” page. We could then show the trend of accidents during a time frame. “View_table” was used to get the whole table in a view statement where users could go in and search the view for an accident by the ID.

We also have stored procedures in our database, “accidents_by_date”, “accidents_by_day”, “accidents_by_state”, “accidents_by_street_side”, “accidents_by_time”, “accidents_by_weather”, “add_full_new_accident”, and “delete_accident”. “Accidents_by_date” gets the table from the view “common_date” and uses a transaction to make sure that there is not a read error. “Accidents_by_state” makes sure that the user enters a state that is in the list. If the user does not enter a state that is in the list, then the transaction rolls back and an error message is given. “Accidents_by_street_side” takes the count of accidents that happen on a street side and returns an error message while not completing the transaction if something went wrong during the read. “Add_full_new_accident” takes in all values that the user wants to input and starts a transaction. If something fails, then the transaction is not committed and an error message is given. “Delete_accident” makes sure the user is deleting an observation from the data set and returns a message if the ID that the user wants to delete is not in the data set.

We created one trigger, “accidents_before_update.” This trigger takes in all the numeric columns and makes sure that the new variables being entered into them during an update are numeric and within the range of the variable. For example, the attribute “start_lat” is the starting latitude coordinate for the accident. Therefore, it can only be in the range of -90 to 90. Thus, our trigger would display an error telling the user that the latitude must be greater than -90 and less than 90

Here is the UML for our database.



The data that was used for testing was from Kaggle. It was created by using APIs to provide streaming traffic incident data and these APIs broadcast traffic data captured by a variety of different entities. We acquired this data by downloading it from Kaggle. There are 4.2 million accidents already in the data set, and users of our application will be able to increase this number by inserting accidents into the data. For users wanting to insert an accident, they will need the identification code and date. If they do not want to use the initialized numbers for severity, starting longitude and latitude, ending longitude and latitude, distance, state, temperature, wind chill, humidity, pressure, wind speed, amenity, bump, crossing, give way, junction, exit, railway, roundabout, station, stop, traffic calming, traffic signal, turning loop, sunrise/sunset, civil/twilight, astronomical/twilight, then the user must provide this information. All empty inputs will be taken as “NULL”. Users can also decrease the number of accidents in the data by

using our application to delete observations from the data. The user will need the name of the attribute and ID of the accident to delete an accident. If the user wants to update an accident in the data, then the user just needs to input the identification code, the attribute, and the new value for the value they are replacing.

In our application, the user will be able to make adjustments to the data like inserting observation, updating observations, and deleting observations. The user may also get the trend of accidents during a time frame, get the distribution of temperatures, get the number of accidents per street side, and get the number of accidents per state.

Illustration of functionality

In order to recreate the database, you first need to download the data. There needs to be a place to store all of the tables for the project so we provide a “CREATE DATABASE” command. You can then use our mega table creation to create a table to store the data and then load the data into the database. The current data is not in third normal form so our code decomposes the data into two tables. Now, the data must be loaded into the decomposed tables. We have created a plethora of views, stored procedures, triggers, and transactions that help keep the integrity of the data. Having loaded these advanced features into the database, you can now switch to the R Shiny code to run and use the application.

The beginning of our code loads necessary libraries in R. The necessary libraries are “shiny,” “DBI,” “odbc,” “RMariaDB,” “tidyverse,” “ggplot2,” and “bslib.” A connection needs to be made to the database so we use “dbConnect()” to do this. This allows us to use our database in mySQL to interact with the Shiny app. Mac users can leave the password as it is; however Windows users will need to remove the password and make it blank. In order to visualize the app, create the “ui” variable which creates the user interface. The other variable that needs to be created is “server” which allows the app to run the commands that you want. Running these created variables in the “shinyApp()” function will create a window in which the app runs.

The browser opens and lands on the home page for our application. We have provided details about the data on this page. At the side of the page are the different user interface pages. The first page gives a look at the first 1000 rows of the data, so users can see the style and attributes of the data. The next page, “Project accidents on map,”

displays the geographical locations of all the accidents in our data. Zooming out to the entire United States shows the areas where information was captured. Zooming in on an area shows the exact coordinates where an accident happened. Curious users can zoom in on their city and even on the street in which they live to see where the accidents are occurring near them. After that is a page is called “Number of accidents by date.” This page will give a line graph of the number of accidents during the chosen date range. The beginning date must be less than the end date and both dates must be in between February 8, 2016 and December 31, 2020. The next page is the “Temperature Histogram.” This page allows the user to view a histogram and the user can determine the number of bins provided in the histogram. After that, the user can visit the “Accidents by Street Side” page where the app provides the total number of accidents on a side of the street. The app also has a page that provides the total number of accidents per state, “Accidents by State.” This page allows the user to choose a state and have the total number of accidents in this time frame in the chosen state returned. Below this returned table is a barplot of accidents by states so users can see where each state compares to other states. The next pages allow the user to interact with the database.

“Search Accident”, “Update Accident”, “Insert Accident,” and “Delete Accident” are the pages that will allow the user to interact with the database. “Search Accident” allows the user to find the observation of a specific ID number and print all of the attributes for that ID. We did this so users could look up an accident and see if anything needed to be changed in the observation. If they needed to update an accident, they could go to the next page, “Update Accident.” This allows the user to update an accident by providing the accident ID, the name of the attribute, and the new value. If the user searches for an accident and it is not there, then they can go to “Insert Accident.” This page allows the user to insert a new accident. The user needs to input an ID for the accident for the insert to work. Otherwise, an error message will tell them that something went wrong and it will not insert the new accident. The numeric variables have been given an initial value with a range for each variable. The user can choose to change the value or keep the initial value. If the user changes the initial value to a blank, then a NULL value will be inserted into the table. The user will not be able to enter any characters into these numeric inputs. The character attributes can be left blank or changed to the values that the user wishes. If the user realizes that they need to delete an accident, then they can go to the “Delete Accident” page. Here the user will be prompted to insert the ID of the

accident. If the accident is in the dataset, then it will return the message, “1 row was deleted.” However, if the ID is not in the data set, then nothing will be returned.

Summary discussion

The status of our application is “complete.” We have completed the necessary items, INSERT, SELECT, DELETE, UPDATE, and we have complete pages that we thought would be interesting to have in our application. The challenging parts of the assignment were creating the application and connecting the database. Creating the application was difficult because neither of us had experience creating a front end application. We overcame this by reading the Rstudio website which gave an overview of the application and some examples of code that we wanted to use. Connecting the database was also difficult because we had to make sure that our code had the right syntax to run in Shiny what we were trying to accomplish in mySQL. Running mySQL syntax in Shiny was difficult when we had errors in our code because it would tell us to check the syntax in our mySQL code without telling us which part of the code was wrong.

Both team members worked diligently on this project and it resulted in a lot of good work being done. Li created the mega table and load statement while Matthew decomposed the table. We figured that this would be the best option because we were still in the early stage of the project and did not want to have burnout of either member. Having both members work early in the project allowed both members to have a say in how the mega table and subsequent tables were created. We both worked on advanced features. This allowed us to create more advanced features than if only one member worked on these. In a large data set, triggers and stored procedures can feel tedious as you must account for almost 50 attributes. Thus, having both members work on these advanced features allowed us to cover a lot of ground. Li created 4 interactive pages and Matthew completed 4 interactive pages. This was another situation where having both members work on the front-end application allowed both members to have a say in how the application would be styled and what features the application should have. It also gave both members the learning experience in creating an application. The concept can be applied to many other data science problems in the future.