CS 3265 Database Management Systems

Project 2 Report - Traffic Accidents Analytics and Visualization of Davidson County

Team Members:
Zhixiang Wang, Zhitao Shu

- Introduction

  This application aims to visualize all of the traffic accidents happened in Nashville for the past 10 years. As two new drivers ourselves, we know from personal experience that Nashville does not have the best road design and drivers, making it difficult to get around the city. Therefore, we want to know how to drive more safely and responsibly by analyzing past accident data and querying useful information. Using the Google Maps API and Nashville Open Data Portal, we are able to plot every single traffic accident in Davidson County in the past ten years onto a map and filter them based on year and month. We also want to know under what weather conditions would it be more dangerous to drive around. We believe with the power of this project, people will be able to learn more about the driving condition in Nashville and become more responsible drivers.

- Implementation
  a. Database side
     1. Data source

        The dataset was acquired from Nashville Open Data Portal. The dataset contains traffic accident records in Davidson County in the past 10 years. The dataset is a csv file.
        Link:
        https://data.nashville.gov/browse?q=Traffic%20Accidents&sortBy=newest&utf8=%E2%9C%93

        The raw data table contains the following fields: Accident_Number, Accident_Date, Accident_Time, Num_Motor_Vehicles, Num_Injuries, Num_Fatalities, Property_Damage, Hit_and_Run , Reporting_Officer , Collision_Type_Code, Collision_Type_Desc, Weather_Code, Weather_Desc, Illumination_Code, Illumination_Desc, Harmful_Code, Harmful_Code_Desc, Street_Address, City, State, ZIP, RPA, Precinct, Latitude, Longitude, Mapped_Location.

     2. Normalization process

        The raw table was in 1NF, where every row represents a single traffic accident. In order to better organize the dataset and simplify the normalization process, a new column was introduced. It is an auto-incrementing variable representing the customed ID (my_ID) of each record. The functional dependencies are as follows:

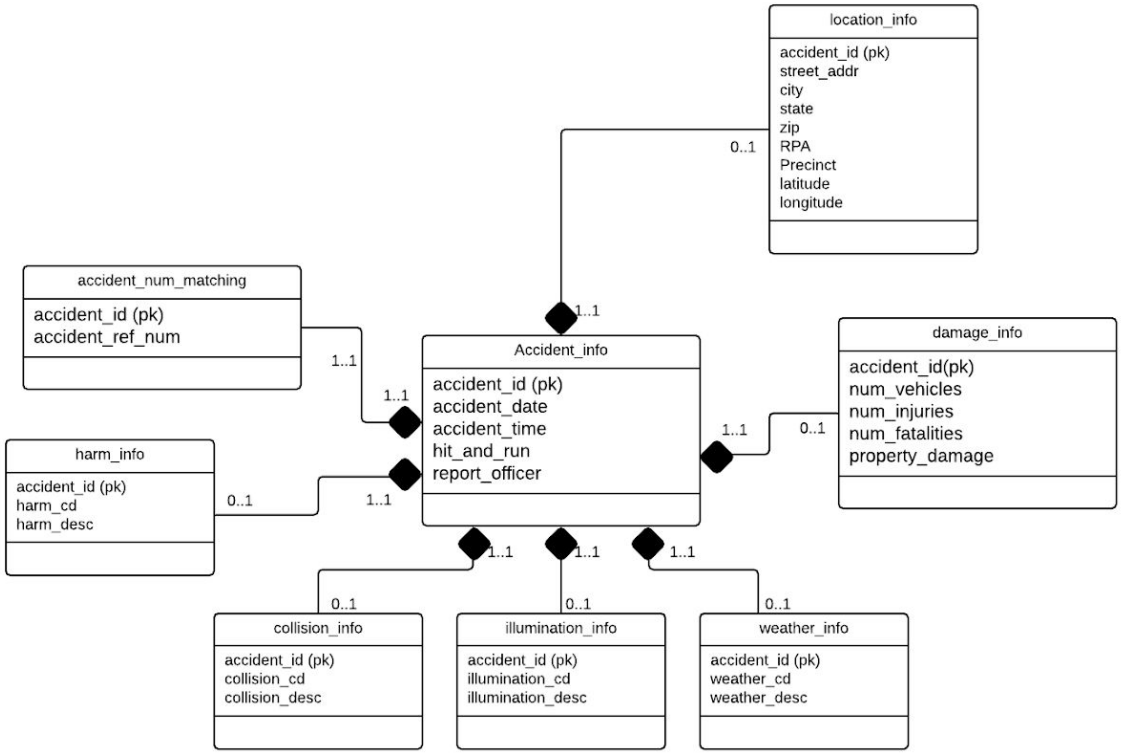
a. my_ID → Accident_Number, Accident_Date, Accident_Time, Num_Motor_Vehicles, Num_Injuries, Num_Fatalities, Property_Damage, Hit_and_Run , Reporting_Officer , Collision_Type_Code, Collision_Type_Desc, Weather_Code, Weather_Desc, Illumination_Code, Illumination_Desc, Harmful_Code, Harmful_Code_Desc, Street_Address, City, State, ZIP, RPA, Precinct, Latitude, Longitude, Mapped_Location.
b. Accident_Number → Accident_Time, Num_Motor_Vehicles, Num_Injuries, Num_Fatalities, Property_Damage, Hit_and_Run, Reporting_Officer, Collision_Type_Code, Collision_Type_Desc, Weather_Code, Weather_Desc, Illumination_Code, Illumination_Desc, Harmful_Code, Harmful_Code_Desc, Street_Address, City, State, ZIP, RPA, Precinct, Latitude, Longitude, Mapped_Location.
c. Collision_Type_Code → Collision_Type_Desc.
d. Weather_Code → Weather_Desc.
e. Illumination_Code → Illumination_Desc.
f. Harmful_Code → Harmful_Code_Desc.
g. Latitude, Longitude → Mapped_Location.

To normalize the table into 2NF, we have to make sure that there is only one primary key. In order to do that, the raw table was decomposed into two tables: one contains my_ID (pk) and Accident_Number, and the other contains my_ID (pk) and all the other fields in the raw table. To further decompose the table into 3NF, all the non-key functional dependencies need to be eliminated. Thus, we need 5 more tables for FD c, d, e, f, g, respectively. To further simplify the tables, the remaining columns were decomposed into several smaller tables. The Mapped_Location column was discarded in the final table design because it is basically latitude combined with longitude.

3. Schema design
The final tables in the database are as follows:
accident_num_matching (accident_id, accident_ref_num), **accident_info**(accident_id, accident_date, accident_time, hit_and_run, report_officer), **damage_info**(accident_id, num_vehicles, num_injuries, num_fatalities, property_damage),

collision_info(accident_id, collision_cd, collision_desc), **illumination_info**(accident_id BIGINT, illumination_cd, illumination_desc), **weather_info**(accident_id, weather_cd, weather_desc), **harm_info**(accident_id, harm_cd, harm_desc), and **location_info**(accident_id, street_addr, city, state, zip, RPA, Precinct, latitude, longitude).

4. UML diagram



b. Web side

1. Front end

❏ Layout

We used D3.js, jQuery, and APIs from Google Maps to build the front end of our web application. jQuery was mainly responsible for constructing and sending the http json request to the back end server and receiving the response json object. The user would be able to choose a certain year and an optional month to see all of the traffic accident locations, a summary of the total number of accidents

categorized by different weather types, and a summary of the total accident statistics. Furthermore, the user would be able to zoom in and out on the map and click on the icon to see the details of an accident.

- ❏ Interaction
  To construct an http request to the server, the application creates a json object consisting of the year and the month values selected by the user from the dropdown menu. After receiving the response from the server, the application would use the metadata to map all the traffic accident locations onto the Google map, construct a bar chart summarizing of the total number of accidents categorized by different weather types, and update the text element corresponding to the total accident statistics.

2. Back end
   - ❏ Design
     We used Node.js and Express.js for the development of the back end web server.
     When the server is started, it will attempt to establish a connection to a mysql database with the specified host, password, port number, and database name (Specifically, localhost, password, 8889, and project2). If the server fails to connect to the database, it will automatically terminate and output the corresponding error message to the console. Once the connection is established, the server will then listen on port 5000 on local host for any http request.
     We used two middlewares to handle the requests: /query/:date and /procedure/:date.
     When the server gets a request to /query/:date, it constructs a query using the values parsed from the date json object which consists of a year and a month. It then commands the database to run the query and sends back the result as a json object to the front end.
     When the server gets a request to /procedure/:date, it commands the database to run the filter_time(year, month) procedure stored on the database server with the values parsed from the date json object. It then sends the result as a json object to the front end.

If an error occurs at any stage, the server automatically shuts down and outputs the corresponding error message to the console.

- Functionality

  In order to run the application, the following requirements must be met:

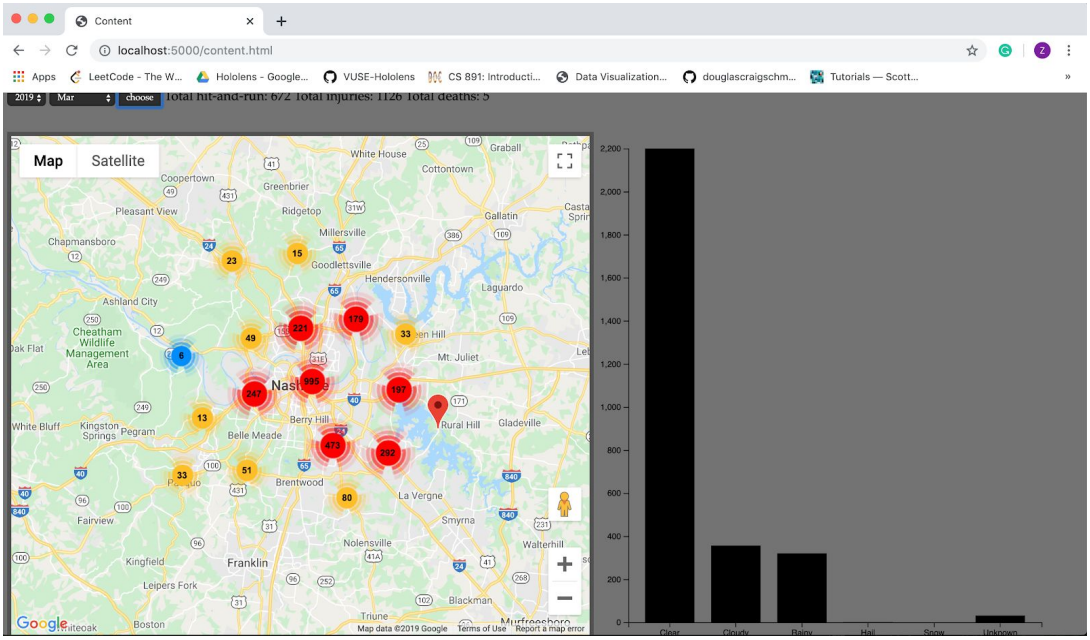
  1. The project2 database must be created and all of the data must be imported into the database. To achieve this, run database.sql in the Database folder and change the path to the raw data if needed.
  2. The stored procedures must be created. To achieve this, run stored_procedure.sql in the Database folder.
  3. The database must be hosted on port 8889 and the password of the root user of the database must be “password”.
  4. Once the database server is up and running, open up a terminal and cd to the BackEnd folder and run “nodemon project”. If the server is started properly, “Connected!” should be outputted to the console.
  5. Open up the browser and go to localhost:5000 and now the web application should be fully functional.

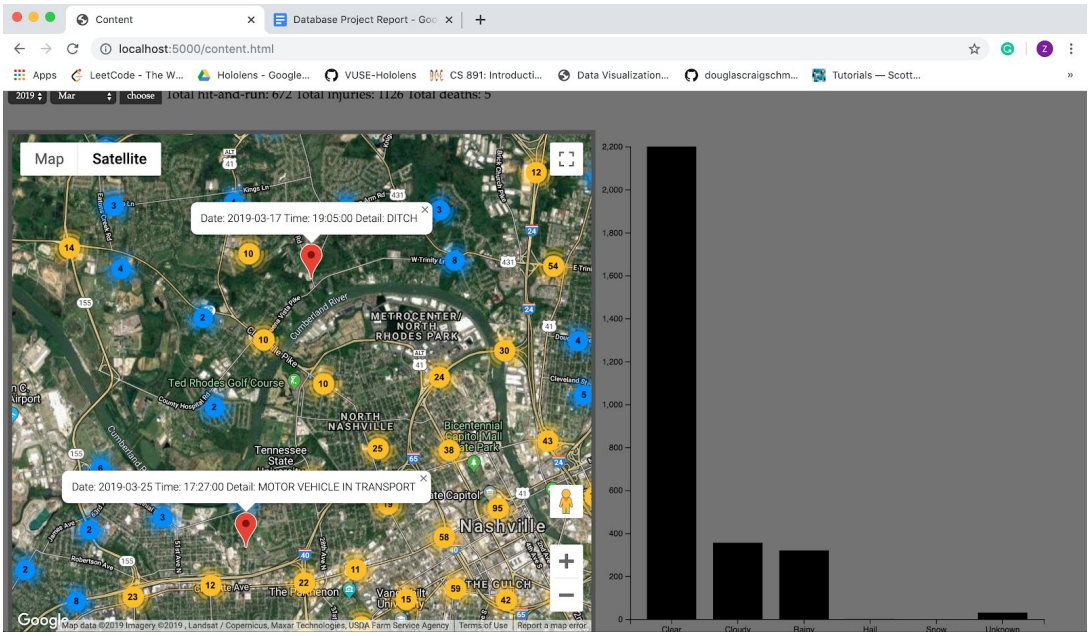
  Once the application is up and running, go to the Content page and use the dropdown menu to select a year and month. Use the middle button of the mouse to zoom in and out on the map. Click on an icon to see the details of an accident.

- Summary
  a. Application status
     - ❏ Database

       All the tables are created and populated using the dataset. Indices are used to speed up queries. Stored procedures are also implemented.
     - ❏ Web

       Everything is functional albeit a lack of aesthetic and robustness. But that is not our main concern for this project.





b. Challenges

- ❏ Database

  The normalization process is quite difficult to visualize, as there are many FDs and attributes. I looked back at the lecture slides and found some useful tips on how to logically decompose a table.

- ❏ Web

  The architecture of the web server is a difficult concept to learn. I watched a lot of tutorial videos on Nodejs and Expressjs and finally understood the route and middleware pattern.

c. Division of responsibilities
   Zhitao Shu: Database design, table decomposition, and stored procedure design
   Zhixiang Wang: Web server and front end architecture and implementation