Project 2 Report

Michelle Zhu and Daniel Shu

Professor Singh

CS 3265 Databases

Accidents in the UK from 2012-2014
Database and application

Introduction

We chose this specific dataset to create our application from because we are interested in drawing practical conclusions from real-world collision data across an expansive period of time. Additionally, we appreciated how specific and descriptive each attribute is – together, they all tell a cohesive story of and give lots of context behind each collision. Some attributes, like numerical day of the week, were clearly included to help users analyze trends in data. With this in mind, we hope to be able to answer questions such as, “Which times of the year have more collisions?”, “Where should we focus road safety efforts?”, and “What are trends in road collisions year-to-year?”. Ultimately, our decision to use this dataset was three-fold: it was expansive enough for us to work with, includes bureaucratic data from another country, and deals with an interesting, day-to-day topic with lots of potential for different types of applications.

We chose to create this application because we wanted to showcase the expansiveness and descriptiveness of the data, summarizing all of the information across three years, as well as provide context for individual accidents. This application can serve as an easy way for travelers to report their own accidents so that others can see the information as quickly as possible. We also want to create insights like road safety at particular junctions, and more dangerous times of the year for road travel. There are lots of opportunities to analyze, and the conclusions drawn from the data have the potential to inform travelers and road safety bureaus to help save future lives. We attempted to tap into this potential with the insights portion of the website.

Final Implementation

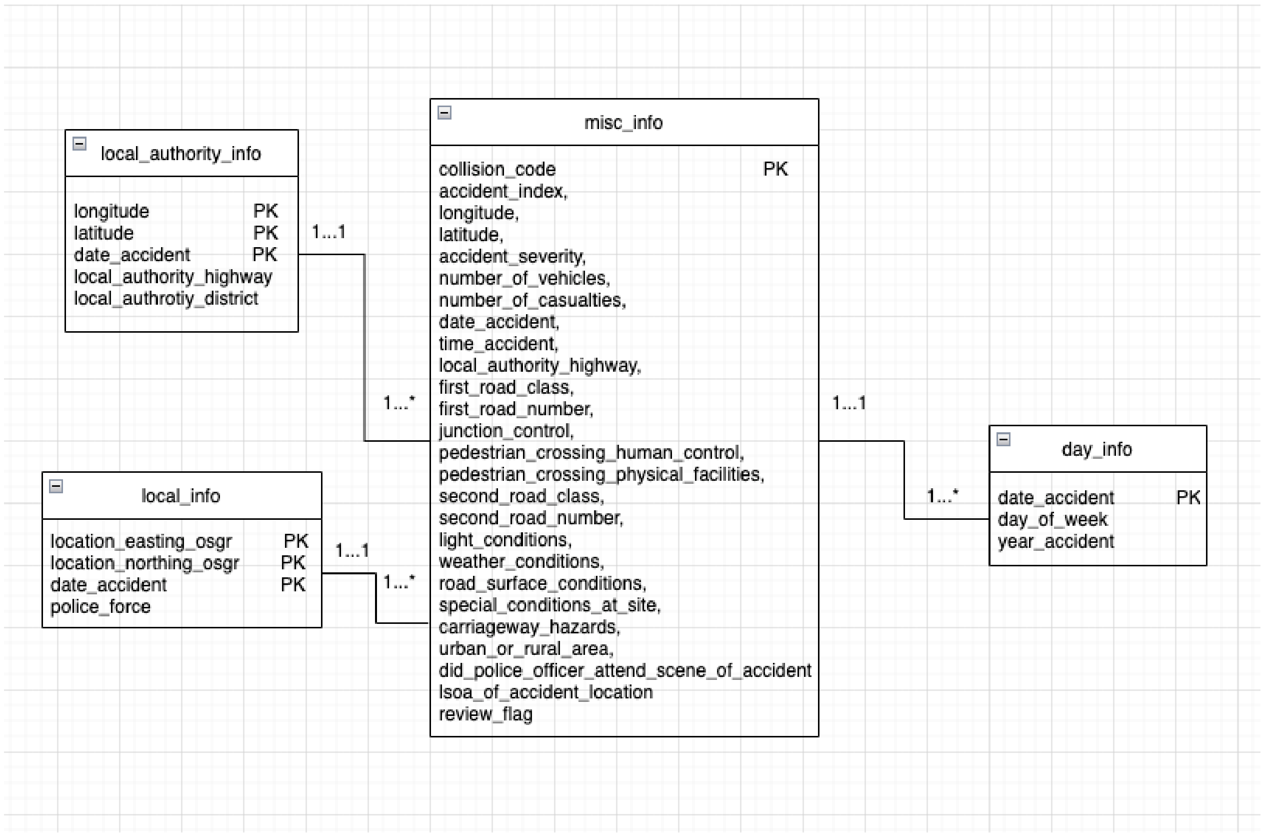
The application is designed for a wide array of users, including road safety experts, witnesses who would like to report and accident, and prospective travelers. The information in the database connects to the front end, offering historical insights, future insights, and the ability to modify entries. The final application contains functions for searching, inserting, and deleting data. It also includes pages for summary statistics and travel danger insights.

First and foremost, we had to work with the database itself so it was ready to use and connect to an application. We started with normalization to split the tables categorically. In this process, we discovered that while many functional dependencies made logical sense, they did not actually hold up in the code. After testing, we identified that day_of_week and year_accident were dependent on date_accident, and that police_force, which is generally divided by geographic location, was based on a combination of location_easting_osgr, location_northing_osgr, and date_accident (date_accident is included because police_force dividing lines shift over time). We separate it into a local_info table, because it gives us information on which police force was responsible for the accident and attending the scene. Additionally, local_authority_highway and local_authority_district are dependent on the same attributes as the previous. We separate it into a local_authority_info table. Therefore, the only functional dependencies we identified that held up in the code are as follows:

Primary Key	**Other attributes**
collision_code	loc_easting, loc_northing, longitude, latitude, accident_severity, number_of_vehicles, [remaining attributes]
date_accident	day_of_week, year_accident
location_easting, location_northing, date_accident	police_force
location_easting, location_northing, date_accident	loc_auth_highway, loc_auth_district

The final database design, therefore consists of four tables. The most expansive and largest table is misc_info, which contains the attributes that are not depending right-hand side of the attributes in the identified functional dependencies. It contains information about the environment of the accident, such as severity, police response, weather, road conditions, and pedestrian presence. The second table is day_info, which includes the day and year of the accident. The third is local_info, which describes which police force had jurisdiction of the accident. Finally, the last is local_authority_info, which describes which highway and district are classified in the accident.

The database design is illustrated in more detail in this UML diagram:



Our initial dataset was taken from Kaggle. For the testing process, we did not use actual data but instead created simple placeholders in the format that real data would take. Then, we employed unit testing to make sure all functionalities were working. This meant we needed at least as many placeholder inputs as there were features of the application, including improper inputs to ensure the application rejected these in a proper manner.

Our application has 5 webpages implementing different features. First, the Summary Statistics page displays various charts showing the percentage of accidents falling under different categories of environmental and circumstantial factors as well as other interesting numbers. Then, we have the travel planner which provides tools that can determine the safety of certain locations or dates based on historical data. It currently implements 2 such tools: an accident propensity calculator, which determines how dangerous a road or an intersection of 2 roads is, and a date safety tool, which determines whether the combination of the inputted date and expected weather conditions are historically safe or not. Next, there is a page for searching the database for accidents by their collision code and a page to report a new accident by entering all the necessary attributes. Finally, we have a page to flag an accident record (identified by collision code) for review or immediately delete it if a user notices any faulty entries.

Illustration of Functionality

The Summary Statistics page requires no input data. It does allow users to click entries in the legends to see the pie charts without them, though the displayed percentages will not change. Note: this page will take up to 15 seconds to load.

The accident propensity index calculator in the travel planner will take in two road numbers that default to 0. After submission, the index will be displayed. A value of less than 0.5 is considered to be safe while one of at least 1 is considered to be dangerous. An example input is 40 for the First Road Number and 1 for the Second Road Number. It returns an index of 0.03, so it is safe.

The date safety tool in the travel planner will take in the planned date of travel and the expected weather conditions during that time. Upon submission, the historical safety will be evaluated. An example input is 01/01/2022 for the date and "Fine with high winds" to be the weather condition. The message "Historically dangerous. Choose another date" is then displayed. If we choose 12/15/2021 for the date, then the message is instead "Historically safe. Good to go".

The search page takes in lower and upper bounds (default to 1) for collision code and outputs all the accidents between them (inclusive). The output table will be magnified upon hovering. Because of memory issues, the max range is 10000. For example, we can enter 10 and 100 for the bounds, and a table will be displayed upon submission showing all rows with collision codes from 10 to 100. If 1 and 10001 are entered, the upper bound will be capped to 10000.

The Report an Accident page allows the user to insert a new accident into the database by submitting values for all the attributes. The only inputs that can be left blank are First Road Number (default 0), Junction Detail (default ''), Second Road Number (default 0), and LSOA of Accident Location (default ''). An example insertion is 1, 1, 1.0, 1.0, 2, 2, 0, 01/01/2012, 01:01 AM, 1, 10, Single Carriageway, 10, none, Authorized person, no physical crossing within 50 meters, 1, 100, Daylight: street light present, fine without high winds , Dry, Roadworks, dislodged vehicle, urban, yes, . A new row will be added to misc_info.

The Faulty Entry Report page allows the user to submit a collision code for two different purposes. If submitted to flag an entry for review, it will update the review_flag attribute of the corresponding row in misc_info to 1, indicating to DB admins that it should be reviewed. If submitted to delete a record, it will directly delete the corresponding row from misc_info. For example, we can input 1 as the code to flag and 2 as the code to delete. The changes will be properly displayed in misc_info.

Summary Discussion

This application is finished. While it is basic, serving as a starting point for a potentially much more complex and expensive application with more pages, more detailed insights, and more intricate UX, it includes all the functions we wanted to implement for this initial step. Some possible extensions:

- Add additional search filters
- Provide support for searching larger selections of data
- Optimize the load time of the Summary Statistics page
- Add data from additional years
- Use machine learning techniques for our travel planner formulas as the ones we are currently using are likely not the most accurate
- Add additional tools to the travel planner

Some challenges we faced were learning how to code the front end and ideating for the application. As we were both very new to creating web applications, learning to use HTML, CSS, and PHP was quite difficult. Even relatively simple tasks such as setting up all the input boxes for the insert function took enormous amounts of time. There were also some interactive and more complex elements we wanted to add such as the charts, so we needed to learn some JavaScript as well. Most of the database, including design and normalization, were straightforward processes, but we struggled and spent much more time finding how to fulfill the requirements in an insightful and compelling way. For example, one idea that we had but did not end up implementing was using a transaction to check validity of attributes from a CSV file to add to the table. Additionally, many of the functional dependencies that we thought should hold, did not actually end up holding when we made the tables. This was solved through copious testing. In the end, few functional dependencies holding was helpful in that the database design was more basic and involved creating fewer tables.

Much of the coding was divided up, whereas discussion, planning, and revising was cooperative. For some checkpoints, Michelle focused on backend work and Daniel did the work on the front end. While the normalization and design processes were time intensive, working on the front end was similarly very time consuming because neither had experience coding in the language. In designing advanced features, Michelle and Daniel worked together to come up with ideas, and split the work of coding and implementing them equally. Much of the project lended itself nicely to a neat division of labor, but for the parts that needed cooperation like design and implementing ideas, discussions were fair with both contributing evenly.