ECE-356 Project | Final Written Report

Tony Shen | Anson Wan | Nikita Kabir
Group 58 | UK Crime

Overview

The goal of our project was to create a simple CLI interface to gain insightful information about UK Crime data. The target audience of our client is policing officials as well as the general public. We leveraged the data from the given UK Crime databases to allow for easy adding, updating, and searching of crimes in the UK depending on the level of authority the user has.

Entity-Relationship Design

The design of our Entity-Relationship Diagram was the first step towards better understanding our datasets. After going through the steps of inspecting our source files (.csv) and loading them into tables inside marmoset, our group decided on the ERD that would provide a structure for our datasets and ultimately provide a mapping towards the construction of our Relational Schema.

Initial Source Files (.csv)

There were 5 initial .csv files that provided all of our source information for this project, spread amongst the 3 Kaggle pages provided for UK crime data:

- https://www.kaggle.com/sohier/london-police-records?select=london-street.csv
    - london-outcomes.csv
        - Outcomes of crimes in London, provides information of a crime's setting, location/jurisdiction, and final outcome
    - london-stop-and-search.csv
        - London stop-and-search data, providing setting, investigation, person, and more information regarding a stop-and-search instance in London
    - london-street.csv
        - Snapshot of current crimes in London, provides same information as london-outcomes.csv (with overlap) but with most recent outcome category and crime type information included
- https://www.kaggle.com/jboysen/london-crime
    - london_crime_by_lsoa.csv
        - Relates the different types of crimes committed to the many LSOAs (local super output region) across London
- https://www.kaggle.com/dabaker/ukregions
    - Output_Area_to_(…).csv
        - Lookup table for various definitions of statistical geography, of which this project only makes use of LSOA (local super output area)

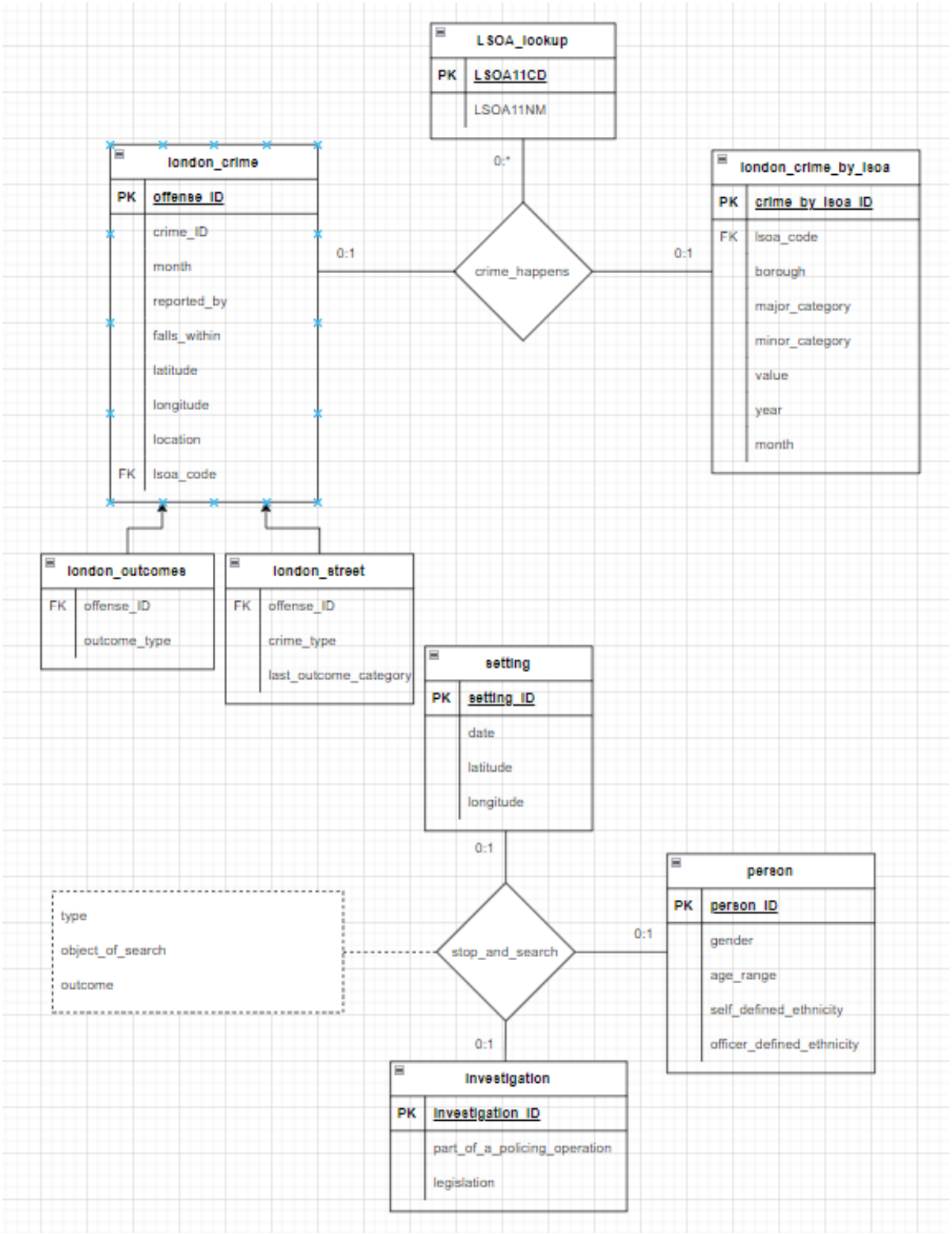
Source Table Setup

Our first step was to load these .csv files into source tables in marmoset04. We mapped these .csv files exactly, as to preserve our original copy of information in an unchanged state. These tables are named the following, and are included in our ERDtoRS.sql document.

- OG_london_outcomes
- OG_london_stop_and_search
- OG_london_street
- OG_london_crime_by_lsoa
- OG_Output_Area

ERD Design Decisions

After multiple trials, we were able to decide on the following ERD to model our Relational Schema. All design decisions are detailed on the next page, describing all of the changes we decided to make from our original (OG) source tables.


LSOA_lookup
PK
LSOA11CD
LSOA11NM
0:*
london_crime
PK
offense_ID
crime_ID
month
reported_by
falls_within
latitude
longitude
location
FK
lsoa_code
0:1
crime_happens
0:1
london_crime_by_lsoa
PK
crime_by_lsoa_ID
FK
lsoa_code
borough
major_category
minor_category
value
year
month
london_outcomes
FK
offense_ID
outcome_type
london_street
FK
offense_ID
crime_type
last_outcome_category
setting
PK
setting_ID
date
latitude
longitude
0:1
type
object_of_search
outcome
stop_and_search
0:1
person
PK
person_ID
gender
age_range
self_defined_ethnicity
officer_defined_ethnicity
0:1
Investigation
PK
Investigation_ID
part_of_a_policing_operation
legislation

london_crime

These fields had many overlapping fields and data entries, as OG_london_outcomes represented crime data in its final state (resolution), while OG_london_street represented a "snapshot" of current crime data (not all resolved). We decided to combine the common fields of these tables into london_crime to save space. We noticed that crime_ID could not be used as a primary key, since there were duplicates which represented multiple stages of a crime, in which last_outcome_category was updated and re-entered into the database. Additionally, we found that instances of "Anti-social behaviour" were not regarded as crimes, and therefore had no associated crime_ID. We decided to add an auto-incremented primary key: offense_ID to facilitate the handling of these duplicate or missing crime_IDs. Further, we took out the LSOA_name column, since we put a foreign key on lsoa_code referencing our lookup table (from OG_Output_Area).

london_outcomes | london_street

With the addition of london_crime, we decided that london_outcomes and london_street would together become an Overlapping Generalization of london_crime. This made sense as a single crime_ID could exist in both tables, since records are not deleted from london_street once they are finalized, instead it is also put into london_outcomes. For each entry, we copied its unique offense_ID from london_crime into either london_outcomes or london_street, depending on which original table the record came from. We then used this as a foreign key referencing offense_ID in london_crime, serving as a link back to all the original information.

london_crime_by_lsoa

The only change made to this table was adding an auto-incremented primary key crime_by_lsoa_ID, which was needed since this information described individual crimes, and thus needed a unique key for each entry to reference to. A foreign key was placed on lsoa_code referencing LSOA_lookup.

LSOA_lookup

As all statistical geography in the other tables in our project dealt only with LSOA (and none of the others), we decided to turn OG_Output_Area into a lookup table for LSOAs (code to name) only. As a part of this process, we were able to delete all duplicate LSOA codes, since this table's only purpose was to provide a mapping from a LSOA code to its corresponding LSOA name.

setting | person | investigation

These fields in each of these 3 tables all come from OG_london_stop_and_search. We decided to split them up into these entities and connect them via a relationship set stop_and_search. We created primary keys on each of these tables using auto-increment. For investigation, NULL entries in part_of_a_policing_operation were assumed to be true (since there were only Falses and NULLs). Also, policing_operation was removed since it was 100% NULL.

stop_and_search

We decided to construct stop_and_search as a relationship set, with the following idea in mind: "stop_and_search ON person AT setting AS A PART OF investigation". A composite primary key was made on the 3 primary keys (setting_ID, person_ID, investigation_ID), with each individual ID set as a foreign key referencing their respective parent table. This stop_and_search relationship set also has descriptive attributes for each instance, also coming from the original table.

Relational Schema

The next step was to translate this ERD into a Relational Schema, using only the source tables loaded into marmoset04 earlier. In general, this simply involved the creation and insertion into a new table, following our ERD. However, there were some extra steps required for some entities/relationship sets, detailed below:

london_crime

As mentioned above, london_crime contains records from both OG_london_outcomes and OG_london_street, with only the common fields between the two. After creating the london_crime table with these attributes only, we first inserted records from OG_london_outcomes, then inserted records from OG_london_street. This would allow us to correctly match records from both london_outcomes and london_street using the auto-incremented offense_ID primary key inside london_crime. More specifically, this was possible since we could use the size of each original table (outcomes, street) to get an exact range of offense_IDs that corresponded to crimes from either OG_london_outcomes (first 1947050) or OG_london_street (last 2946479). However, after verifying the total number of distinct offense_IDs (= 1947050+2946479 = 4893529) was correct, we noticed a gap of 19000 between the two "blocks" (outcomes, street). After some research, we attributed it to a known behaviour of mysql auto-increment when executing multiple bulk insert intos: https://dba.stackexchange.com/questions/60295/why-does-auto-increment-jumps-by-more-than-the-number-of-rows-inserted

As the entire table had already been loaded, we decided to hardcode the difference of 19000 into our offense_ID field. We knew that offense_ID = 1947050 represented the last "outcome", and knew that the first "street" would have offense_ID = 1947050 + 19000 + 1 = 1966051. This was done through manually setting auto-increment to start at 1966051 in london_street. By doing this, we were able to map crimes from london_crime correctly to london_outcomes and london_street, keeping all information connected through the matching offense_ID.

With this implementation, we had to consider how our client app function of adding a crime would be affected. Our approach for adding crimes was to add to london_crime first, then take the max(offense_ID) in london_crime (representing the crime just added) to be put into either london_outcomes or london_street. Even though offense_ID in london_outcomes and london_street are by default auto-incremented, we learned that we could manually provide these values so that they would always match the correct entry in london_crime.

london_outcomes | london_street

For london_outcomes, we only needed to add the field Outcome_type from OG_london_outcomes, since all other fields were already in london_crime (and linked through FK on offense_ID, see above). Similarly, london_street required the addition of only Crime_type and Last_outcome_category from OG_london_street. The only difference for london_street was manually starting its auto-increment to account for the gap in london_crime, as per above.

london_crime_by_lsoa

The creation of this table was very simple, as all attributes from OG_london_crime_by_lsoa were inserted. As mentioned, an auto-incremented primary key was added to allow for all entries to be unique.

LSOA_lookup
The creation of this table was also very simple, as only LSOA11CD and LSOA11NM were inserted. Insert Ignore was used to handle the duplicate LSOAs, since we did not need any of the other (more geographically accurate) fields that shared the same LSOAs. After this table was created we added the foreign keys on lsoa_code from both london_crime and london_crime_by_lsoa to reference LSOA11CD in LSOA_lookup, allowing those tables to lookup the correct LSOA names.

setting | person | investigation
The creation of each of these 3 tables were simple, and involved inserting only the specific fields (from OG_london_stop_and_search) required by each entity as per the ERD. An auto-incremented primary key was created for each, which is both referenced from the below stop_and_search relationship set, and part of its composite primary key.

stop_and_search
Creating the stop_and_search relationship set involved the insertion of every single field from OG_london_stop_and_search, including the descriptive attributes not included in setting, person, or investigation. Foreign keys were added on setting_ID, person_ID, and investigation_ID referencing each respective table, and a composite primary key composed of all 3.

Client Application

Given our final ERD and relational schema, our client implementation includes three core functionalities: Adding Crime Entries, Updating Crime Entries, and Searching Crime Entries.

Add Crime Entries
Adding a crime is inserting an entry into either london_crime and london_outcomes (DONE crime), london_crime_by_lsoa, or stop_and_search and its related entities. Since london_outcomes is a specialization of london_crime, when we insert into london_crime, we automatically generate a new primary key offense_ID. We then copy this key and assign it to london_outcomes (see above). The same is true during update when inserting into london_street (see below). Adding is restricted to police official user types only.

Update crime entries
Updating a crime is inserting an entry into london_crime and london_street (UPDATED crime status). One original idea was to edit entries directly when updating, however upon closer inspection of our given tables, we noticed that updates are already handled by inserting into london_street. Also, editing entries directly would require additional security support to track history of edits by certain users. Such functionality is ideal, but is out of the scope of this project. Updating is restricted to police official user types only.

Search crime entries
Search desired tables based on the user's specified attributes. To perform a search, a user first selects which table to search from a menu, then selects which attribute(s) to filter by. Filtering prompts a user to input a string of digits, where each digit corresponds to a certain attribute for the

chosen table. Multiple of the same digit in the string is accepted to allow for filtering by multiple values of the same attribute. Searching is available to all user types.

List top crimes given an LSOA

The user will be prompted for an LSOA value and how many entries to list. The program will then list the crimes and their occurrences, sorted by occurence and limited to the specified amount of entries. This feature is available to all users.

List top LSOAs sorted by crime frequency

The user will be prompted for how many entries to list. The program will then list the LSOAs and occurrences of crimes within those LSOAs, sorted by occurence and limited to the specified amount of entries. This functionality is available to all user types.

UI Design Decisions

Initial menu immediately prompts the user for their role which in turn is their authorization level. Reason being that the functions the user will have access to is directly related to their authorization level since a normal everyday citizen should not be able to add crimes or update crimes to the database. As such, if the user is a citizen they will only be allowed to search, list top crime-types, and list highest crime LSOA's. Throughout the client there are many enumerated menu's used to navigate and select values for certain attributes. This design choice was to prevent the user from searching for attribute values that will not return any results back since after looking at the initial data set, many attributes do not have unique values and are only allowed certain values. Also within the client are menus where the user is specified values to input themselves (rather than an enumerated list) this is the case for when a user is adding and updating mostly, but for some fields when searching as well. We decided to use these menus when the values to be inputted are either unique (i.e: crime_id) or the possible values are too high of a number to list (i.e: year-month).

Functional Design Choices

Updating Entries

- Updating entries directly was decided to not be supported because of security risks such as not knowing who updated the entry and why it was updated. We thought updating was particularly interesting when discussing this as a group because of the various ways one can implement it and the changes that would need to be accounted for to support editing such as adding an edit receipt so we know that an entry was edited. We thought about possibly adding an "Edited" and "Reason for edit" attribute column to solve the edit receipts problem, but since the majority of the fields will end up being empty or null (since most of the entries will not be edited), we decided on a different approach. We ended up deciding to make updating an entry simply an add by leveraging the last-outcome-category attribute in the london-street table. So essentially, the london-street table contains all crimes AND their updates, where the update is done by adding the crime again with the same crime_id but now it has a different value in the last-outcome-category attribute. This way, one can easily see the edits by searching for a certain crime_id and looking at all the last-outcome-category columns to see how the crime "went through the pipeline" so to speak, until it resolves.

Deleting Entries

- Deleting entries was intentionally neglected as functionality because of possible security risks. Even if a user is a policing official, there is no way to delete an entry as crimes should not be removed from the database since a crime is a crime after all. If an entry was accidentally added for any reason, we again leverage the last-outcome-category column in london-street.

List top crimes by LSOA

- List top crimes given an LSOA was decided to be a standalone function in the client as it is an important query that can give insights to policing officials about what crimes they may need to crack down in certain areas of the UK.

List LSOAs by crime frequency and infrequency

- List top LSOAs by crime frequency or crime infrequency was also decided to be a standalone function as a citizen may want to know which areas in the UK are the highest in crime for various reasons such as when settling down to purchase a home or to start a family.

Ideal Client vs. Actual Implementation

An ideal client would include additional features such as:

- A nice GUI, rather than the primitive program purely through console.
- More security support to keep track of which users are performing which actions. Currently, we simply ask the user whether they are a police official who needs extra privileges, or a standard citizen user. We also do not verify their identity, so anybody could simply select that they are a police official to easily get increased access. Ideally, users would have unique logins and would verify those logins with passwords upon running the program.
- Edit entry and delete entry, with some level of restoring recently edited or deleted entries. This would be made possible by the additional security support in the above point. Currently we do not support these features at all.

Test Case Plan

Most of our testing for the Client Application involved first executing the client code from an end user's perspective, then verifying that the changes to our database reflected the expected behaviour (using manual SQL queries in backend). Some examples for each of our functionalities are listed below, with details regarding the exact test case (from the user) alongside the corresponding code (backend) executed to verify functionality. Some of the test cases are included in our code demonstration video, so please follow along for a live demonstration.

Add crime entries

Test Case #1 (london_outcomes):

- Added a crime to london_outcomes, which entails first adding the general fields to london_crime, copying the newly auto-incremented offense_ID from this table, and then inserting offense_ID and outcome_type to london_outcomes
    - Multiple crimes were added in tests to ensure that the system responded correctly to more than one user add

- To verify: we checked that each of the newly added crimes was correctly placed into and linked together between both london_crime and london_outcomes. Most importantly, we made sure that the offense_IDs matched between the two tables for the same entry added.

Test Case #2 (stop_and_search):

- Similar process, checking that the crime was added to setting, person, then investigation (with consistent ID), then finally added to stop_and_search

Update crime entries

Test Case #3: (london_street)

- Updating a crime is the same as adding it to london_street (new last_outcome_type), so this test case followed a similar process: first adding to london_crime and ensuring that the data was all there and correct, then letting the client application add the remaining 2 fields to london_street and ensuring that the offense_ID matched for the two added entries

Search crime entries

Test Case #4: (london_outcomes)

- Searches london_outcomes for crimes occurring in 2014 June, in LSOA_codes E01032739 and E01032740, with outcome_type = “Investigation complete, no suspect identified”
- Verified by consulting original .csv file (apply filters on columns), and also running query manually in backend

Test Case #5: (stop_and_search)

- Searches stop_and_search for stop and searches conducted on Males 18-34, with self_defined_ethnicity = White, searching for Controlled drugs, and the offender was given a drugs possession warning
- Verified by consulting original .csv file (apply filters on columns), and also running query manually in backend

List top crimes given an LSOA

Test Case #6: (london_crime_by_lsoa)

- Tested using test LSOA E01001043, with limit 10 entries
- Verified by consulting original .csv file (apply filters on columns) and Kaggle, and also running query manually in backend

List top LSOAs sorted by crime frequency (most/least)

Test Case #7: (london_crime_by_lsoa)

- Tested highest # crimes with limit 15, then lowest # crimes with limit 20
- Verified by consulting original .csv file (apply filters on columns) and Kaggle, and also running query manually in backend