

**Advanced Databases/Databases Technologies 2022/2023**

**Project Report**

Inês Fragoso 54454, Robert Morosan 54456, Milosz Wloch 59437, Martim Silva 51304

**Description of how to replicate the project creation**

For each of the four files, one should run each individual set of cells from top to bottom with the separation of the infrastructures involved in this project, SQLite, and MongoDB, being signalled by the file name beginning with SQLite or MongoDB respectively and the order to run these files is as follows:

1. *sqlite\_create\_db.pynb*, *mongodb\_create\_db.pynb* -> Used for database creation and data cleaning/insertion into each database;
2. *sqlite\_queries.pynb*, *mongodb\_queries.pynb* -> Used for query creation and also optimization coupled with the implementation of adequate indexes for each database.

**Note**: The original data files (**source**: https://www.kaggle.com/datasets/kwullum/fatal-police-shootings-in-the-us?select=PoliceKillingsUS.csv) should be kept in a folder titled “archive” in the root directory where all the *Jupyter Notebook* files are located and without alterations to their titles and contents.

**Description of the dataset**

This dataset is a compilation done by the American newspaper "The Washington Post" since the start of 2015 up until around 2017 of some of all of the US police shootings that resulted in casualties. The information gathered includes the race, age, gender of the person who was fatally shot, if the person was carrying a weapon or not, when/where this shooting took place, whether the victim showed signs of mental illness, if the police officer had a camera on their person at the time, among other pertinent information about each shooting. This dataset also contains information regarding the median household income per each city; the percentage of people over the age of 25 that have completed high school per each city; the poverty rate per each city and the shares of people of a given race (from white, black, Hispanic, Asian, and native American) per each city. All of this information was originally extracted from the United States Census.

**Scheme used for both databases**

It is worth noting that in this project, it was decided to keep the same database schemas for both types of infrastructures, the diagram used to represent them is based on the Unified Modelling Language standard and is presented below.

Diagram, schematic

Description automatically generated

**Note**: For SQLite implementation the *PoliceKillings* id field is auto incremental despite the existence of an *id* field in the PoliceKillings.csv file

**Discussion of point done/not done in the project**

1. For the first goal of the project, we selected a dataset in the Kaggle website that was the "Fatal Police Shootings in the US" by Karolina Wullum. We decided to select this dataset over others as it provides a big quantity of relatively clean data, more specifically a small amount of missing/mismatched data. Also this topic provides interesting information upon its further exploratory analysis. It is worth mentioning that every CSV file must have at least one column that is somehow linked to one another (this allows the setting of foreign keys when converting these files to collections). Anyway, before we could make use of this data by implementing a database, we had to create some constraints as primary and foreign keys to relate them. A UML diagram was created to present attributes and relations between tables. Some additional tables have been added to enable better data maintenance in the future.
2. For the second goal of the project, we were expected to create the database for the relational infrastructure in SQLite and the NoSQL infrastructure in MongoDB using Python and the libraries studied in the practical classes, all while maintaining the overall structure of the data itself set in the diagram that was previously built. In order to import the CSV files into databases the data had to undergo through prior processing using Data Frames from Pandas, this included changing the string values of the various races into a single format across all CSVs, replacing recurring, but not universal, ambiguous suffixes of city names (namely "town", "city", "CDP") that would result in some discrepancies when connecting the data between tables/collections. Further cleaning was performed, like in the State table/collection where we operated with the unique values that the file PercentOver25CompletedHighSchool.csv provides for the states and we also mapped the corresponding ids from collection to collection in the MongoDB database according to the scheme used and shown in the scheme image where a collection (in this particular instance) would be connected to another by sharing its id field with it thus creating a foreign key in this second collection, despite the fact that we are not really using them that way here since MongoDB is a NoSQL database and therefore doesn’t use the relational model.
3. For the third goal of the project, we were supposed to create at least six different queries for each of the databases. An important detail to underline here is that each and every one of the queries needed to be “equal” to the corresponding one performed in the other database. We created in total 7 queries, 3 simple queries which selected data from one or two columns, 2 complex queries that used joins and aggregates and one update and insert query as well. The update and the insert detail, respectively, updating the date of an instance of a police killing and inserting a new instance of a police killing. Below there is a table with every query implemented and its output which is the same on both databases.

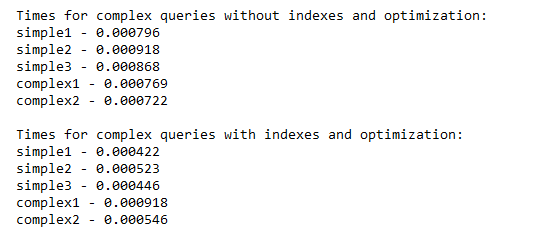
|  | **Simple Query 1** | **Simple Query 2** | **Simple Query 3** | **Complex Query 1** | **Complex Query 2** |
| --- | --- | --- | --- | --- | --- |
| **Query Details** | Unarmed Males under 18 killed by Police | Women killed by Police while not fleeing and not armed | People with NULL values regarding age, race and whether they were armed or not | People above the age of 20 armed with a knife who were shot and tasered by Police in the state of Texas | State with the most shootings against males under the age of 18 |
| **Result Table for the query in both DBs** | [('Jeremy Mardis', 6),  ('Jordan Edwards', 15),  ('Jose Raul Cruz', 16),  ('Deven Guilford', 17),  ('Armando Garcia-Muro', 17),  ('David Joseph', 17)] | [('Ciara Meyer', 12),  ('Alteria Woods', 21),  ('India Kager', 28),  ('Autumn Steele', 34),  ('Justine Damond', 40)] | [('TK TK', 'M', '16-11-11')] | ('Randall Lance Hughes', 48, 'TX', 'knife')  ('Henry Reyna', 49, 'TX', 'knife')  ('Gregory Mathis', 36, 'TX', 'knife')  ('Ray Valdez', 55, 'TX', 'knife')  ('Rodney Henderson', 48, 'TX', 'knife') | {'\_id': 'CA', 'shootings': 11} |

1. For the fourth and final goal of the project, we were trying to make the queries we have done before to be faster by applying indexes for specific attributes of the databases. For the SQLite part, we created three indexes of every possible type in SQLite.

Created indexes:

* **Single field index:**
* state\_id (**city** table)
* **Unique index:**
* acronym (**state** table)
* **Compound index:**
* city\_id, armed, gender, age (**police\_killings** table)

We decided to create a single field index for attribute state\_id in the city table as it was the one field that mattered in our JOIN instructions. Unique index was done for acronyms as values of this field cannot be repeated because there is only one state with specific acronym. We created a compound index regarding four fields such as: city\_id, armed, gender, age because these attributes were present in WHERE and JOIN clauses in our queries. After indexes were created, we tried to optimise our complex queries. In the case of the first complex query we changed our LEFT JOIN clauses into INNER JOIN. In the case of the second query we replaced the subquery by JOIN instructions and made it less composite. At the very end of the last goal of the project we made a comparison of the performance of queries with and without indexes based on returned time that was calculated as a mean for 20 iterations so that it could be more reliable. In SQLite we were not able to make our query use our indexes for the third simple query and the first complex query, so the score of them should not be considered when comparing these two times. In the below output results are taken from optimised queries for complex 1 and complex 2 queries. From the screenshot we



#TODO

also make a plot with the times mongo vs sqlite