UNIVERSITÀ DI PISA

Decision Support System

A.Y. 2023/2024

Project Assignments

Part 1

Part 2

Part 3

Manai Eva 257280

Andriani Paolo 579604

Velardita Michele 578770

Part 1

1. Database Schema

During the creation of the relational schema, by replicating the data warehouse schema of reference, it was chosen to remove the *Incident* table as it is a degenerate dimension (it does not contain any other columns besides the primary key).

About the type chosen to represent data, we believe that the following types can best fit the needs of a correct simple representation: to *custody_id, incident_id*, and *date_id* fields have been assigned the type 'int', and for the missing ids (p*articipant_id, gun_id* and *geo_id*) the 'varchar' type was assigned, as they were subsequently generated as strings (see paragraph 3).

2. Retrieval of geographic information

The Python script *get_locations_googleMap.py* starts by importing necessary libraries such as *googlemaps* for the Google Maps API, *csv* for handling CSV files, and *alive_progress* for a progress bar. It then initializes a Google Maps API client using our personal API key.

The script defines a function named *reverse_geocode* to extract city and state information from geographical coordinates iterating over the Google Maps API response.

Next, it reads coordinates from the *Police.csv* file and removes duplicate coordinate to avoid the same reverse geocoding request to google API. The script proceeds to write the results to a new CSV file (*city_state.csv*). The output CSV file includes columns for latitude, longitude, city, and state.

Error handling is implemented within the *reverse_geocode* function to catch and print exceptions that may occur during the API request.

In summary, the script efficiently processes coordinates from a CSV file, performs reverse geocoding using the Google Maps API, and writes the obtained city and state information to a new CSV file. It includes a progress bar for visual tracking of execution progress and handles potential errors during the geocoding process.

The creation of the geo id is done in a separate script. This decision was taken because the choice of how to build the geo id had not yet been taken at the time of retrieving location information, and because the request to the API of every coordinate take a long time to perform, we preferred to keep the two operations separate and modifying the data afterwards.

The script *create_geo_id.py* starts reading the coordinate from the file previously created (*city_state.csv*), the geo id is created by concatenating the latitude and longitude values and, from the resulting string, removing the point and the minus character. This because the resulting string is more compact and still maintain the uniqueness for every record, this is given by the fact that the data concern only US, so the latitude will always be negative and thus, removing the minus from it, does not compromise the uniqueness of the key (for example in the database it is not possible to have this two coordinates 41.7527,-87.6338 and 41.7527,87.6338 that will result in the same surrogate key because one is in US and the other is in China). The result of this operation is then saved in a new csv file (*city_state_id.csv*)

3. Generation of missing IDs

Starting from given data, *Police.csv* file, the three *.json* dictionaries and the *dates.xml* file, it was produced the following information in string type, needed to construct the database schema required: first we generate the participant's ids using this format:

<participant_gender><participant_age_group><participant_status>< participant_type>

So, for example, from the record:

`192533,Teen12-17,Male,Arrested,Suspect,41.5643,-90.5975,Stolen,Shotgun,105332,812`

We obtain the synthetic information *"`M321`"* simply concatenating available information: it means that the incident refers to a male participant (by taking only the first letter of the gender in the available string), that he is a teen (12-17 y.o.), and that he is considered suspect and arrested.

Second, we build geographical ids concatenating again in a single string the two available values of latitude and longitude without the point and the minus character (they are originally expressed in decimal degrees (DD) format) obtaining the string *<latitude><longitude>.* So, for example, from the record above we get the string *"`415643905975`".*

Finally, we obtain the gun_id as concatenation of given values in <*gun_type*>_<*gun_stolen*> string, obtaining from the same previous sample record the string *"`Shotgun_Stolen`".*

This kind of encoding allows us to define different categories of participants, without the need to store data of individuals involved in the incident, or about the incident itself, in the database tables. The same for guns and locations: we are building prototypes of categories to better generalize characteristics of entities needed.

4. Loading records into tables

The process of loading records into the tables is facilitated by the *SplitContent.py* file. This file orchestrates the loading procedure by defining multiple functions responsible for extracting pertinent information not readily available in the files.

The initiation of the data loading process involves the extraction of information from *dates.xml, Police.csv*, and *city_state_id.csv*. For each line parsed from these files, after the extraction and computation of missing data, the *'build query'* function assembles the string that represents the INSERT query. The function receives as inputs the table's name, a list of column names, and a corresponding list of values to be inserted. Each value is formatted before generating the final string, which is then passed as input into the *perform query* function. This function executes the query, commits the changes, and manages potential errors that may arise, specifically addressing common *IntegrityErrors*, such as the insertion of a duplicate key or a violation of a foreign key constraint.

In summary, for each file, the process of loading data into tables follows the pipeline:

- Read data from files.
  - Extract data line by line, excluding any header information.
  - Loading the attributes into local variables.
- Compute missing data.
  - Use utility functions to calculate missing ids, supplementary date values, and the gravity of crime.
- Build query string.
  - Create a list of columns names.
  - Format values, enclosing string in quotes, converting empty string into 'Null' and converting non-string values into string.
  - Construct the query string:

  ```
  INSERT INTO {table_name}({columns_string}) VALUES ({values_string})
  ```

- Perform query.
  - Execute the query generated before while catching possible exceptions.

Some implementation choices are reported below:

- Insertion order: inserting a record containing a foreign key whose primary key is not yet present in the database violates a system constraint (raising an IntegrityError). It was decided to avoid the modification of the database constraints when inserting the records and to execute the queries in the correct order, so as not to raise the exception, giving priority to the dimensional tables.
- 'Unknown' values in *gun_stolen* field: the occurrence of 'Unknown' values in the 'gun_stolen' field is preserved, as they are considered valid potential values and are not transformed into 'Null'.
- Clean Tables: an executable code (*>python CleanTables.py TableName1 TableName2...*) has been developed to delete the records loaded in the specified tables. If there are issues with data integrity, foreign key violations, or other errors during the data loading process, *CleanTables.py* provides a way to selectively delete records from specified tables and re-execute the data loading process.

Part 2

The strategies adopted and the SSIS nodes used to resolve the different assignments are described below. Every result is written in a csv file using a **Flat file Destination** node.

0. Assignment 0: *<For every year, compute the number of total custodies.>*

First, we select from the Custody fact table the custody_id, and add the column Year through a **lookup** over Date table. Then, with an **aggregation** node, the records were grouped by Year, counting the number of custody_id for each group.

1. Assignment 1: <*A state is considered to have a youth criminal problem if the age group with the highest overall gravity consists of individuals under 18. List every state with a youth criminal problem.*>

First, the columns of interest were selected from the Custody fact table: custody_id, participant_id, geo_id and crime severity. Using the **lookup** nodes, age_group and state columns have been added from the Participant and Geography tables. Through the **derived column**, the age_group column has been modified changing the values in adult (18+ years) and young; subsequently, with an **aggregation** node the records were grouped by age_group and state, performing a sum on all crime_gravity. In order to compare the crime_gravity of the two age groups, the records were divided into two dataflows using a **conditional split** node with age_group as condition, **sorted** on the state and finally combined via **merge join**(inner join on state column). In this way it was possible to have the state, the crime gravity of young people and the crime gravity of adults on the same row. After this, through a **derived column** node, a fourth boolean column was added representing the result of the comparison "young crime_gravity">= "adult crime_gravity".

2. Assignment 2: <*For each incident, compute the ratio between the total gravity of crimes with a stolen gun and the total gravity of crimes with a not-stolen gun.*>

First, we retrieve the columns: gun_id, incident_id and crime_gravity from the custody table, after that we add the column is_stolen from the Gun table with the **lookup** node taking the gun_id for the lookup. After that, we split the data through the **conditional split** based on the value of is_stolen and **aggregate** on the incident_id and summing up the crime_gravity, having, at the end, for each incident the sum of the crime gravity. This operation is done for both data flows (stolen and non-stolen guns). After this, we merge the two dataflows with a **merge join**. We chose to do a left join on the not-stolen data because, for how the ratio is defined (total gravity stolen/ total gravity not-stolen) it will be undefined if the denominator is zero, so doing a left join, we allow the stolen gravity to be zero but not the not-stolen one (we do not take in consideration incidents were is not present a non-stolen gun). In the end we change the null values of crime gravity stolen, resulting from the previous left join, to zero and calculate the ratio of the two.

Part 3

0. Assignment 0: *<Build a datacube from the data of the tables in your database, defining the appropriate hierarchies. Create the needed measures based on the queries you need to answer.>*

Data cube construction

For answering the queries, we decided to build a cube with *4* dimensions hierarchy: ***custody,*** ***date***, ***gun***, and ***participant***. In *custody* the dimensions used were *custody_id*, *incident_id* and *state*, the latter is retrieved from the relation of the Custody table with Geography. *date* contains *date_id* and *year* dimensions without hierarchies, as we only need the year for the first query. *Gun* contains *gun_id*, *gun_type* and *is_stolen*; *gun_type* is used for the second query and *is_stolen* is used by the PowerBI dashboard. *participant* contains *participant_id* and *age_group*, used in the first query, and *gender*, used in the dashboard.

The only measure used in the queries is *crime_gravity* already present in the fact table. We created a new calculated measure used in the PowerBI dashboard, the *gun id count*, calculated counting all the non-empty values of *gun_id* from the custody table. We used this measure together with the *is_stolen* dimension for analyzing the stolen and not stolen gun ratio for each state.

1. Assignment 1: *<Show the percentage increase or decrease in total crime gravity with respect to the previous year for each state.>*

MDX Query

To calculate the percentage increase between one year and the next, two members were defined: the first (diff), using the *currentmembe*r and *prevmember* functions, calculates the difference between the crime_gravity of one year and the previous one; the second (diff_percent) calculates the percentage variation using the previously defined member.

The query returns the absolute difference and the percentage change between subsequent years, for each *state*. To avoid infinite values in the *diff_percent* column, an if was executed in the declaration of this last member (in case the denominator was zero, a zero will be returned.

2. Assignment 2: *<For each gun, show the total crime gravity in percentage with respect to the total crime gravity of all the guns.>*

To calculate the percentage of crime gravity for each gun, compared to the total, two members were declared: *total,* with the use of the parent function on the *gun_type* attribute, calculates the total crime gravity on all the guns; *gun_percent*, on the other hand, calculates the percentage ratio between the *crime_gravity* on each individual gun and the total.

The query returns, sorted in descending order, each type of gun with the measure previously defined.

3. Assignment 3*: <Show the incidents having a total gravity score greater or equal to the average gravity score in each state.>*

Given the high computational calculation required by this query, as mentioned before in the "Data Cube construction" section, the *state* attribute was inserted within the *Custody* dimension hierarchy also containing the *incidents*. Since, in this way, the calculation of the average does not require the join of two dimensions on the part of the OLAP system, execution is much faster.

Before declaring the query that answers the business question, another MDX was created that returns the average crime gravity for each *state*. The average was calculated through a calculated member *avg_crime*, which uses the default function *avg*. For each *state*, the average of the incident ids belonging to that *state* is then returned.

To return the incident id whose crime severity is greater than or equal to the average of the state to which it belongs, the filter function was used on the tuple (*State, Incident*). The nonempty function was also used to exclude the set of empty tuples.

4. Assignment 4: *< Create a dashboard that shows the geographical distribution of the total crime gravity in each age group.>*

Dashboards

Geographical distribution of Crime Gravity

As starting point (and mandatory assignment) we plot the geographical distribution of *crime_gravity* per *state* and *age_group* using PowerBI map and bar plot in which we see that per-age crime gravity is somehow related to the geographical location stored in *state*. In this bar plot, we observe how among different states, the proportion of adults and teens crime gravity is pretty the same, with some little exceptions. So, *state* seem to do not impact so much in changing the ratio between adults and teens involved in a crime. What really changes is the interval in which the value of *crime_gravity* lies, highlighting that some states are more prone to have dangerous situations than other. This can be due to demographical factors as density of population and presence of big cities, or maybe to geographical collocations, as the proximity to the coasts.

Then, we use a line plot to see, over the years *2013-2018,* how *crime_gravity* vary with respect to two categories in *age_group*, teens and adults. We decided to remove the *age_group* "child" of *0-11* y.o. because of its low frequency in data. We notice how the two distributions both follow an increase over the years, where the adult's one is only steeper

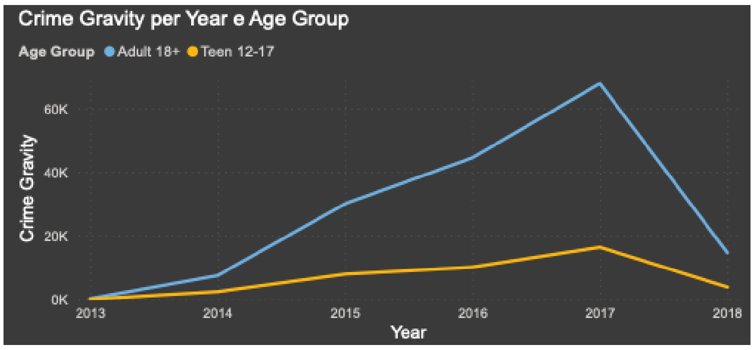

Figure 1: Crime gravity increases over years

than the other. We believe that the peak in *2017* has no real importance, given that data collected in *2018* are incomplete (they regard only the first three months).

5. Assignment 5: *<Create a dashboard that shows the geographical distribution of the total crime gravity in each age group.>*

Geographical distribution of Guns and Crime Gravity analysis per genre

In the second page of PowerBI report, we want to analyze two main aspects: if there is some kind of genre difference while considering the *crime_gravity* value, and if the ratio between *stolen* and *not-stolen* guns per *state* changes over *years*.

As expected from literature, *males* have strongly higher values of it, meaning that more *males* than *females* are involved in crimes, but also observing that the impact is much higher. Also, the usage of guns in crimes has a genre-based difference: usually females have other kind of preferences as also stated here: [FBI crime report U.S.]()

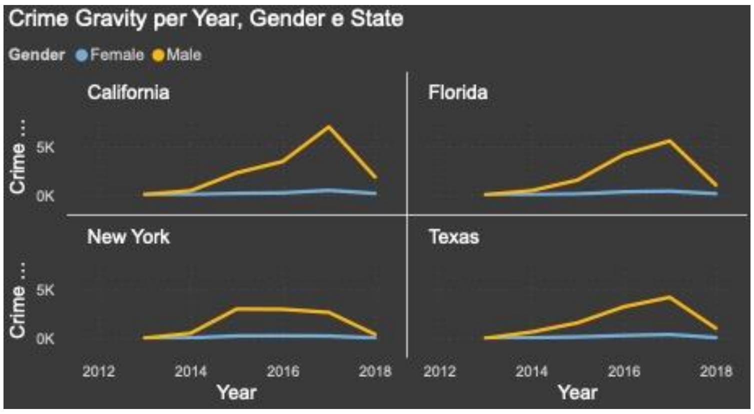

Figure 2: Crime gravity over years increase more for males. Here there are the four states with higher overall crime gravity

We can also guess other directions in which drive our research, by raising other hypothesis, for example that the lower value on females means also that they're more often victims (that led to zero the whole *crime_gravity* computation).

Then, for *crime_gravity*, we want to observe if in the four states with highest values there is some kind of trend over the years, for both males and females. We notice that male distribution follows better what have been observed previously in *Figure 1*.

Now, focusing on *gun_Id_count*, we want to see if there are some patterns among states, and between *stolen* and *not_stolen* guns. We observe that most crimes are realized with stolen weapons, and only a small percentage with not stolen ones. This ratio between the two categories appears stable among all the states, and thinking as humans to these results, we can easily guess that possession of illegal weapons is linked to an increase in crimes.

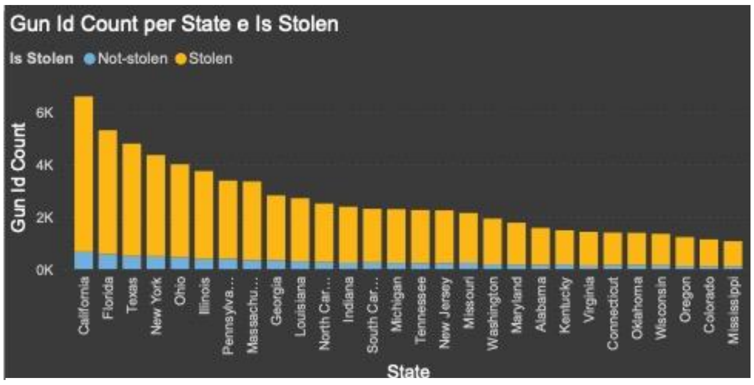

Figure 3: Number of crimes in which a gun is used. Stolen guns are prevalent across all states

Ultimately, we observe the total guns used in crimes along the time interval considered in the dataset. We notice that the distribution is very similar to the one in *figure 1*, so we can see that the increase of usage of stolen guns along years, as shown in *figure 4*, also correspond to the increase of crime gravity in the same period, where stolen guns follow the same distribution of *crime_gravity* in adults and not stolen guns the same distribution of *crime_gravity* in teens.

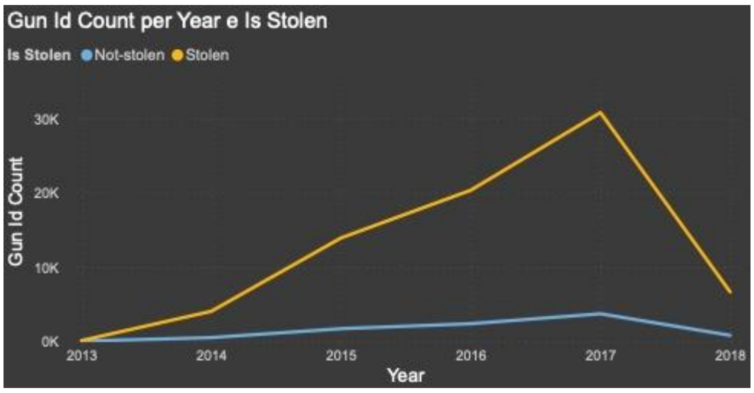

Figure 4: Crimes in which a stolen gun is used has a steep increase over years