

LABORATORY OF DATA SCIENCE

POLICE PROJECT

December 29, 2023

Andrea Cardia

Università di Pisa

Contents

0	Introduction	1
1	Construction of the Data Mart	1
1.1	Assignment 0: Creation of the Schema	2
1.2	Assignment 1: Data Extraction and Transformation	3
1.3	Assignment 2: Data Loading	3
2	Querying the Data Mart	4
2.1	Assignment 0: SSIS Query 1	4
2.2	Assignment 1: SSIS Query 2	5
2.3	Assignment 2: SSIS Query 3	6
3	Multidimensional Analysis	8
3.1	Assignment 0: Creation and deployment of the Cube	8
3.2	Assignment 1: SSIS Query 1	8
3.3	Assignment 2: SSIS Query 2	9
3.4	Assignment 3: SSIS Query 3	9
3.5	Assignment 4: Power BI Dashboard 1	10
3.6	Assignment 5: Power BI Dashboard 2	10
	Appendix A: SQL Solutions of Queries of Section 2	11
A.1:	Assignment 0 Part 2	11
A.2:	Assignment 1 of Part 2	11
A.3:	Assignment 2 of Part 2	13

0 Introduction

In this document, we will outline how to build a data warehouse on **Microsoft SQL Server** from a dataset related to legal cases. The initial dataset consists of a table with 170,928 rows, each corresponding to a different custody. The columns of this table contain information about:

- Case identifier: an attribute that uniquely identifies each of the 170,928 custodies.
- Age of the participant of the custody: this attribute can take three different values representing age ranges: 0 to 11 years, 12 to 17 years, and over 18 years.
- Gender of the participant: an attribute with 2 possible values (male or female) identifying the gender of the participant.
- Status of the participant: an attribute with 4 possible values (arrested, killed, injured, or unharmed).
- Type of participant: an attribute that can be either suspect or victim.
- Latitude: indicates the latitude at which the incident occurred.
- Longitude: indicates the longitude at which the incident occurred.
- Gun stolen: provides information about whether the weapon used in the incident was stolen or not (it can also be unknown or irrelevant for a total of 4 different distinct values).
- Type of weapon: contains one of 28 possible values about the type of weapon used in the incident.
- Incident identifier: an identifier for the incident. In the table of 170,928 rows, there are 105,168 distinct values for this attribute; this means that an incident may refer to multiple custodies simultaneously.
- Date identifier: contains information about the date on which the incident occurred.

Additionally, beyond the table, the dataset also included three dictionaries containing information about the correspondence of values for participants' age, participants' status, and participants' type. In these dictionaries, each possible value of these three attributes is mapped to a number used to calculate the primary measure of the data mart that will be constructed. This measure is the severity of a crime and is determined by:

$$crime_gravity(x) = F_1(x.participant_age) * F_2(x.participant_type) * F_3(x.participant_status) \quad (1)$$

Another file contained a table with information about the dates of the incidents. This document was used to map the integer values in the table to actual dates.

In the first part (also referred to hereinafter as 'section') of this document, we will delve into the ETL processes, namely the stages of data Extraction, Transformation, and Loading onto a server¹. In the final stage, we will create an actual database, or more precisely, since the data at hand consists of aggregated historical data, a data mart, which is a specific type of data warehouse with a single fact table. Once the data mart is created, in the second part, we will proceed to query it by exploring different solutions with various softwares such as **Microsoft SSIS**, **SQL**, and **Extended SQL**. Finally, in the third part, after constructing the OLAP cube with **Microsoft SSAS**, we will address additional analysis queries using the **MDX** language, a structured language designed specifically for exploring data warehouse hierarchies.

1 Construction of the Data Mart

Before the data can be distributed, the database needs to be created in its schema; this is the skeleton of the database. It is therefore essential to create the correct tables with the right attributes (i.e., their data types) and the connections between them through the mechanism of foreign keys. Poor definitions at this stage can lead to errors during data loading. For example, a data that is defined as a certain type in the schema will be accepted by the database only if it is of that type. In subsection §1.1, pertaining to Assignment 0, we will illustrate how this schema was created by using **Microsoft SQL Server Management Studio (SSMS)**. In subsection §1.2, corresponding to Assignment 1, we will discuss the operations carried out regarding data extraction and transformation. Finally, in the discussion of Assignment 3 in the first part, namely in §1.3, we will conclude the section by examining the last phase of the ETL process, namely the data loading phase.

¹For all these operations, we will use **python** with its basic functions, without relying on well-known libraries such as **pandas**.

1.1 Assignment 0: Creation of the Schema

In Figure 1, the schema of the Custody table is shown. Here, the only attribute that can have null values is the crime.gravity measure, which will be calculated later. All other attributes, being foreign keys (or the primary key in the case of custody_id, as indicated by the symbol to its left), cannot accept null values. It was decided to construct these attributes as integer numbers. While it is natural to think of foreign keys and primary keys as integer values, the choice of data type for the crime.gravity measure is not as straightforward. As we will see in the following sections, calculations such as percentages or ratios will be performed on this variable. Therefore, it is not advisable for this attribute to be an integer. Creating this attribute as an integer would require casting for ratio and percentage calculations, resulting in computational resource expenditure. However, even though it would be more appropriate to define it as real values for these reasons, it was decided to keep it as integer values to explore the casting functions of Microsoft SSIS and Microsoft SSAS, respectively, in Sections 2 and 3 of this document.

Column Name	Data Type	Allow Nulls
date_id	int	<input type="checkbox"/>
day	nvarchar(50)	<input checked="" type="checkbox"/>
day_of_the_week	nchar(10)	<input checked="" type="checkbox"/>
month	nvarchar(50)	<input checked="" type="checkbox"/>
month_of_the_year	nchar(50)	<input checked="" type="checkbox"/>
year	nvarchar(50)	<input checked="" type="checkbox"/>
quarter_of_the_year	nvarchar(50)	<input checked="" type="checkbox"/>
quarter	nchar(50)	<input checked="" type="checkbox"/>

Figure 2: Date Table in Microsoft SSMS

in this context and refer you to the attached `ipynb` files. In Figure 2, the schema of the Date Table is shown.

In Figure 3, the schema of the Geography Table is shown. This table has an integer primary key, latitude and longitude attributes as real numbers, and city, country, and state attributes. The only attribute that does not accept null values is the primary key. This key allows connecting to the corresponding attribute in the Custody fact table of the star schema, acting as a foreign key. In the following subsection, we will illustrate how the attributes city, country, and state were created based on latitude and longitude data².

Column Name	Data Type	Allow Nulls
gun_id	int	<input type="checkbox"/>
is_stolen	nchar(10)	<input checked="" type="checkbox"/>
gun_type	nchar(15)	<input checked="" type="checkbox"/>

Figure 4: Gun Table in Microsoft SSMS

In Figure 5, the schema of the Participant Table is shown. This table contains a primary key (a non-null attribute) and the 4 attributes referring to the participants in the incident. Determining the primary key will be the subject of a specific analysis in the following sections, as the initial dataset did not provide this information. As usual, it plays the role of connecting to the corresponding foreign key in Custody.

Column Name	Data Type	Allow Nulls
custody_id	int	<input type="checkbox"/>
incident_id	int	<input type="checkbox"/>
participant_id	int	<input type="checkbox"/>
gun_id	int	<input type="checkbox"/>
geo_id	int	<input type="checkbox"/>
date_id	int	<input type="checkbox"/>
crime_gravity	int	<input checked="" type="checkbox"/>

Figure 1: Custody Table in Microsoft SSMS

The Date Table has the date in the year-month-day format as its primary key and contains seven other attributes representing the year, quarter, month, day, and their respective versions contextualized within what will be revealed as a hierarchy of fundamental importance. This table is connected to the Custody fact table through its primary key (which appears as a foreign key in Custody). The main challenge in creating this table was reading the date file in `xml` format. For brevity, we do not present the `python` code used for this operation

Column Name	Data Type	Allow Nulls
geo_id	int	<input type="checkbox"/>
latitude	real	<input checked="" type="checkbox"/>
longitude	real	<input checked="" type="checkbox"/>
city	nchar(80)	<input checked="" type="checkbox"/>
state	nchar(30)	<input checked="" type="checkbox"/>
county	nchar(70)	<input checked="" type="checkbox"/>

Figure 3: Geography Table in Microsoft SSMS

In Figure 4, the schema of the Gun Table is shown. This table includes a primary key and two attributes derived from the initial dataset table. We will see later that extracting the primary key for this table will require interpretive effort³. Here too, the primary key connects the table to the Fact Table.

Column Name	Data Type	Allow Nulls
participant_id	int	<input type="checkbox"/>
age_group	nchar(10)	<input checked="" type="checkbox"/>
gender	nchar(10)	<input checked="" type="checkbox"/>
status	nchar(10)	<input checked="" type="checkbox"/>
type	nchar(10)	<input checked="" type="checkbox"/>

Figure 5: Participant Table in Microsoft SSMS

²We will explore how it is possible to employ various approaches to solve this problem, considering two, in particular, and analyzing their advantages and disadvantages.

³Here, as with the Participant Table, it will be necessary to define more specifically what an element of the table represents. We defer these discussions to Section §1.3.

1.2 Assignment 1: Data Extraction and Transformation

After creating the data mart schema in **Microsoft SSMS**, it was necessary to break down the initial dataset to organize it for populating the schema tables. Here, we focus on the process that led to the extraction of geographical data from latitude and longitude. In this regard, two **python** libraries were compared: **geopy** and **reverse_geocoder**. Both, given a pair of coordinates (latitude, longitude), generate a dictionary containing various information about the geographical location in question. **geopy** distinguishes cities into boroughs, villages, towns, and cities, while **reverse_geocoder** does not, listing them all under the same 'name' entry. Therefore, in this respect, it requires less attention. Some geographical locations corresponding to cities without a county (such as Saint Louis, Missouri) are correctly classified as 'N/D' by both libraries. However, to remove this missing value, these entries will be replaced with the city name. Another difference between the libraries is the number of classes and functions to call to obtain the desired result. In the case of **geopy**, it is necessary to instantiate a **Nominatim** object, pass the coordinates to the **reverse** method, and then extract the relevant information from the dictionary. On the other hand, **reverse_geocoder**, given a tuple containing tuples corresponding to the coordinates to be mapped, directly provides the dictionary. Additionally, in the case of **geopy**, two exceptions had to be handled to prevent the program from freezing during server requests. Undoubtedly, the main difference between the two libraries lies in the time required to obtain the correct mapping. For **geopy**, due to the limited number of requests allowed to the server (2/3 requests per second), the mapping time for the 95,655 distinct coordinate pairs was 13/14 hours. For the **reverse_geocoder** library, since the coordinates are already stored in memory, the mapping time is negligible. A comparison between the mappings resulting from the two methods also suggests that the mapping done with **reverse_geocoder** is even more accurate than that done with **geopy**.

1.3 Assignment 2: Data Loading

Data loading was performed using the **pyodbc** library in **python**. The data loading procedure first requires establishing a connection using the connection string. Once the connection is established, it is possible, for each table, to iterate through all the rows of the table and load them one by one.

- Loading the Date Table: When loading the Date Table, the **Datetime** library was used to find the correspondence between a date and its day of the week. For each row in the **dates** list, the values of the respective columns to be inserted in the corresponding table of **SSMS** were extracted, and an insert query was executed.
- Loading the Participant Table: The Participant Table was created by simply taking the Cartesian product of the four lists, each containing the possible values of the various attributes of the Participant Table⁴. Thus, having 3 distinct values for the age of the participant, 2 distinct values for gender, 4 distinct values for status, and 2 distinct values for type, what was obtained is a table with 48 rows. A different participant is, therefore, in this interpretation, a particular combination of these 4 attributes, rather than, as one might otherwise understand, a specific person. This approach was chosen due to the impossibility of distinguishing between one person and another. It would have been possible for the same person to commit two or more offenses, but with the available data, it is infeasible to distinguish between such individuals. The loading of the table was done with **pyodbc** by executing 48 INSERT queries.
- Loading the Gun Table: For the Gun Table, it was decided to proceed with the same logic used for constructing the Participant Table. Instead of understanding a different weapon as a specific object with a unique serial number, it was decided to interpret it as a specific combination of the available attributes. With only two attributes (excluding the primary key) available, with 4 and 28 distinct values respectively, a table with 112 rows was obtained. The table was loaded as usual by executing an INSERT query for each of these rows with the **pyodbc** library.
- Loading the Geography Table: As discussed earlier, the rows of the Geography Table correspond to the number of distinct positions (latitude-longitude pairs) in the dataset. For each of the 95,655 rows, an INSERT query was executed to insert the data into **SSMS**.
- Loading the Custody Table: Two methodologies were explored for creating the Custody table: in the first, a search was conducted on the lists of other tables for each combination of attributes in a row of the initial dataset to find the corresponding key. This methodology proved to be immensely time-consuming. To address this, the data structures of the tables were changed from lists of dictionaries to dictionaries where each combination of attributes corresponded to the primary key. This way, every search could occur in constant time, and the creation of the entire Custody table took only a couple of seconds.

⁴Clearly, the primary key was not considered in the four attributes; the primary key was created incrementally by assigning an integer to each element of the Cartesian product.

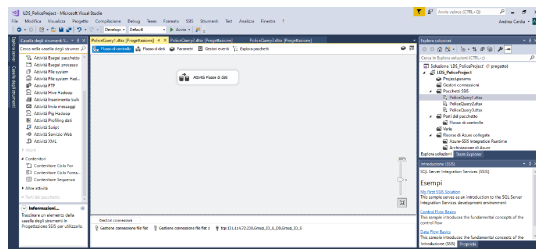
2 Querying the Data Mart

In this second phase, we will test the data mart built in the first phase by executing 3 queries. In each of the following subsections, these queries will be described and analyzed, and their resolution will be done using Microsoft SSIS. For a resolution of the same queries through the SQL language and the Extended SQL language, please refer to Appendix A.

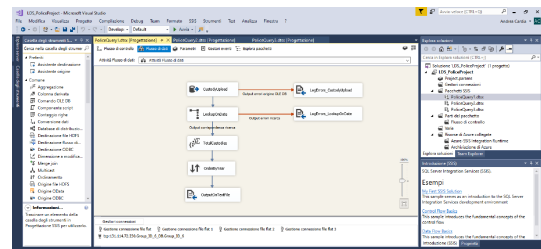
2.1 Assignment 0: SSIS Query 1

Query. *For every year, list the participants ordered by the total number of custodies.*

To execute this query in Microsoft SSIS, after creating a new package within a new project, a DATA FLOW TASK node was created (in the Control Flow window) where all the operations characteristic of the database querying process will be executed. In Figure 6a, you can observe the Microsoft SSIS interface in this initial phase.



(a) Control Flow of Query 1



(b) Data Flow of Query 1

Figure 6: Solution of Query 1 in SSIS

On the right, among the various packages that make up the entire project containing the solutions of all the queries we will discuss in this section, the one concerning the solution of the query in question is highlighted under the PoliceQuery1.dtsx; On the left, you can see some of the possible operations, each of which has a node (a button) associated with it, executable in Microsoft SSIS; Finally, in the center, there is the diagram constituting the Control Flow of the query in question.

Once a DATA FLOW TASK node is inserted, double-clicking on it allows configuration. This operation takes us directly to the Data Flow of the package⁵. In Figure 6b, there is a screenshot of the Data Flow for the first Query. This diagram consists of a set of nodes that together allow, starting from the data warehouse built in Section 1, to solve the proposed query.

Here is a bulleted list describing the various nodes of the process:

- **CustodyUpload:** This is an OLE DB SOURCE node; it allows connecting to the server where the Data Warehouse is present and loading the Custody table into the process. Only the columns of interest, namely participant_id and date_id, are taken from the Custody Table. The error output, i.e., what the node should do in case of an error in the loading phase, is configured to redirect error-causing rows; the destination for these rows is determined by the LogErrors_CustodyUpload node connected to the right of the Custody Upload node via a red arrow. By configuring this node, the relevant rows are redirected to a text file named LogErrors_Query1.CustodyUpload.txt.
- **LookupOnDate:** This is a LOOKUP node; it allows, starting from table A, to perform a JOIN with table B on key K, provided that table B does not have rows with the same value for key K. In this way, table B is used as a hash table, and the search for matches occurs in constant time⁶. In our case, table A is Custody, table B is Date, and clearly the key K is date_id. We only take the year column from the latter, which is what we need to solve the query. Here too, the error output is configured to redirect error-causing rows to the text file LogErrors_Query1.LookupOnDate.txt via the LogErrors.LookupOnDate node.
- **TotalCustodies:** This is an AGGREGATION node; it allows performing what in SQL is the GROUP BY operation. So, in our case, grouping must be done by year and participant_id, and on this grouping,

	A	B	C
1	year	participant_id	TotalCustodies
2	2013	29	1
3	2013	0	2
4	2013	33	2
5	2013	39	2
6	2013	42	2
7	2013	4	3
8	2013	12	3
9	2013	45	3
10	2013	16	4
11	2013	31	4
12	2013	24	5
13	2013	8	7
14	2013	20	7
15	2013	41	11
16	2013	28	12

Figure 7: Result of Query 1

⁵With the term "Package" in Microsoft SSIS, we mean that set of operations that characterize ETL processes from a source to a destination, data cleaning, complex data transformations, and the automation of integration processes.

⁶At the expense, of course, of a greater expense due to the construction of the hash table.

custodies must be counted, i.e., rows (since each row is associated with a custody). This count is renamed TotalCustodies, and the output columns of this node are naturally the two of the grouping and the one resulting from the count. Error handling is not provided for nodes of this type.

- **OrderByYear:** This is a SORTING node; it allows performing what in SQL is the ORDER BY operation. In our case, sorting (for the reason explained earlier in the resolution of the SQL query) is done on the three attributes first by year, then by participant_id, and finally by the TotalCustodies measure. Error handling is not provided for this node either.
- **OutputOnTextFile:** This is a FLAT FILE DESTINATION node; it allows writing the contents of the input table (i.e., in our case, the result of the query) to a file. The chosen name for the output file is OutputQuery1.txt, and the format is csv. Figure 7 displays the resulting file opened with Microsoft Excel. Error handling is not provided for this node either.

2.2 Assignment 1: SSIS Query 2

Query. For each state, compute the stolen gravity index defined as the ratio between the total gravity of custodies involving stolen guns divided by the overall gravity of custodies.

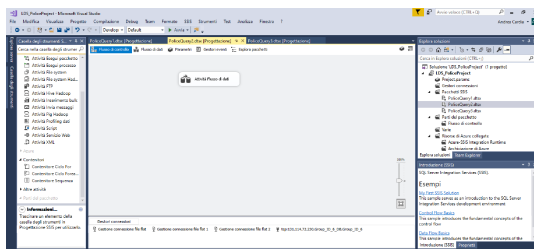
To execute this query in Microsoft SSIS, the same logic as the second SQL solution⁷ was followed.

Firstly, a new package was added to the existing project, and within the Control Flow, a Data Flow Task was inserted. Figure 9a shows a screenshot of this phase. By double-clicking on it, you enter the Data Flow depicted in Figure 9b. Here begins the process of extracting data from the Datamart, processing and transforming it, with the final result written to a text file. This process is characterized by the following nodes:

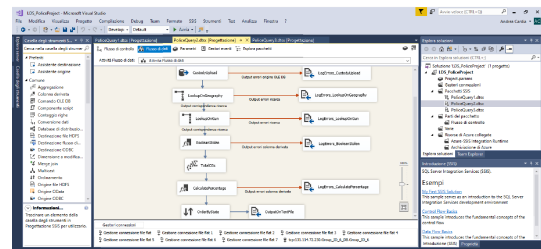
- **CustodyUpload:** This is an OLE DB SOURCE node; it accesses the Custody table of the Datamart, taking the columns useful for JOIN operations (i.e., the foreign keys gun_id and geo_id) and the column related to the crime_gravity measure. Error output is configured to redirect error-causing rows to a text file named LogErrors_Query2.CustodyUpload.txt, managed by the LogErrors_CustodyUpload button on the left.

	A	B
1	state	StolenGravityIndex
2	District of Columbia	0.775
3	Maine	0.7845
4	New Hampshire	0.78758
5	Mississippi	0.78825
6	Missouri	0.78945
7	Kentucky	0.79356
8	Alaska	0.79887
9	Connecticut	0.80006
10	Pennsylvania	0.80185
11	Arkansas	0.80194
12	South Carolina	0.80205
13	North Carolina	0.80372
14	Arizona	0.80433
15	Georgia	0.80513
16	Montana	0.80556

Figure 8: Result of Query 2



(a) Control Flow of Query 2



(b) Data Flow of Query 2

Figure 9: Solution of Query 2 in SSIS

- **LookupOnGeography:** This is a LOOKUP node; This node performs the JOIN with the Geography table. The use of the LOOKUP node rather than the MERGE JOIN is preferred as it is less computationally complex (see the discussion of LookupOnDate in Section 2.1). The node is configured to take the state attribute from the Geography table, and error output is configured to redirect error-causing rows to the LogErrors_Query2.LookupOnGeography.txt text file through the LogErrors_LookupOnGeography button.
- **LookupOnGun:** This is a LOOKUP node; Its purpose is to do exactly the same as the node discussed earlier, with the difference of doing it on the Gun table using the gun_id attribute as the hash key (remembering that the attribute by which the JOIN is performed is nothing more than the key used in the hash table built by the LOOKUP), and from there, taking the is_stolen attribute.

⁷Please refer to Appendix A.2

- **BooleanStolen:** This is a DERIVED COLUMN node; Here, a new column is created using the following expression:

```
(TRIM(is_stolen) == "Stolen") ? (DT_DECIMAL,15)crime_gravity : 0.00
```

The `? :` operator represents the conditional construct; it evaluates the expression to the left of the `?` symbol and returns the value to the left of the `:` symbol if the evaluation is true, while returning the value to the right of the `:` symbol if the evaluation is false. In our case, the expression to be evaluated is that the field related to the `is_stolen` attribute⁸ is equal to the string "Stolen". If this evaluation returns the true logical value, the field related to the new column being created for the row being analyzed is `crime_gravity`⁹ and 0¹⁰ otherwise. The newly created column is finally renamed as `BooleanStolen`, and error output redirects rows causing errors to a dedicated text file.

- **TotalCGs:** This is an AGGREGATION node; Here, grouping is done by state, and for each group - therefore, for each state - the sum is made of all values related to the `crime_gravity` measure (this sum is renamed `TotalCG`) and of all values related to the `crime_gravity` measure but corresponding to rows with `is_stolen = 'Stolen'` (this sum is renamed `TotalCGStolen`).
- **CalculatePercentage:** This is a DERIVED COLUMN node; Here, a new column is created using the following expression:

```
(DT_DECIMAL,5)ROUND(TotalCGStolen / TotalCG,5)
```

Thus, for each state, the ratio of the values of total `crime_gravity` restricted to stolen gun custodies and total `crime_gravity` is calculated. The `(DT_DECIMAL,5)` cast is applied to eliminate all non-significant zeros. The obtained ratio is renamed `StolenGravityIndex`, and error output is configured as usual, redirecting problematic rows to a text file.

- **OrderByState:** This is a SORT node; This node sorts the rows of the resulting table to display them in descending order with respect to the `StolenGravityIndex` measure.
- **OutputOnTextFile:** This is a FLAT FILE DESTINATION node; Here, the resulting table is written to the `OutputQuery2.txt` text file in csv format. Figure 8 displays the resulting file opened with Microsoft Excel.

2.3 Assignment 2: SSIS Query 3

Query. *For each month, compute the total gravity in percentage with respect to the annual total.*

In Figures 11a and 11b, the initial and final parts of the data flow diagram are respectively shown. The control flow consists, as for the previously resolved queries, of a single node related to a single data flow task. For the sake of brevity, the corresponding figure has been omitted. As done previously, we will proceed with a point-by-point analysis of the data flow diagram.

- **CustodyUpload:** This is the usual OLE DB SOURCE node that connects to the Datamart and loads the Custody table into the SSIS project. The columns taken from the Custody table are `date_id` and `crime_gravity`. Error output has been configured as usual, redirecting rows with problems to a text file in csv format.
- **LookupOnDate:** This is the LOOKUP node that performs the INNER JOIN on the `date_id` key with the Date table. The columns taken from this table are year and month. Error output redirects rows to a text file in csv format.

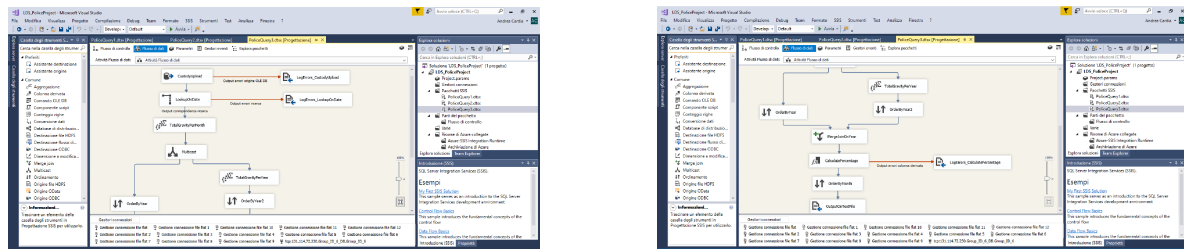
	A	B
1	month	TotalGravityPercentage
2	2013-01	6.67
3	2013-02	8.63
4	2013-03	14.9
5	2013-04	9.02
6	2013-05	11.76
7	2013-06	9.8
8	2013-07	7.06
9	2013-08	7.84
10	2013-09	11.37
11	2013-10	5.49
12	2013-11	3.92
13	2013-12	3.53
14	2014-01	6.48
15	2014-02	4.68
16	2014-03	7.08

Figure 10: Result of Query 3

⁸Stripped of spaces at the beginning and end of the string being analyzed by applying the TRIM function to it

⁹The `(DT_DECIMAL,15)` function in Microsoft SSIS casts to a float number with 15 digits after the decimal point of the variable to its right. This transformation is necessary since we will later have to perform a division that, if done between integers, will result in 0 in all cases where the numerator is less than the denominator.

¹⁰Here, we have written 0.00 to ensure that the result is not an integer but a floating-point number.



(a) First part of the Data Flow of Query 3

(b) Second part of the Data Flow of Query 3

Figure 11: Solution of Query 3 in SSIS

- **TotalGravityPerMonth:** This is an AGGREGATION node; here the total crime_gravity is calculated at the granularity of month. Rows are then grouped by year and month, and the sum of the crime_gravity measure is calculated on this grouping.
- **Multicast:** This is a MULTICAST node; the function of this node is very simple. It copies the input table into different copies (two in our case), splitting the data flow into two branches where identical tables "flow" in each branch. This node does not have the option to configure error output.
- **OrderByYear:** This is a SORT node; this node is placed along the left branch of the split performed by the MULTICAST node and sorts the table by year in anticipation of the MERGE JOIN operation, which requires the tables to be merged to be sorted according to the same criteria.
- **TotalGravityPerYear:** This is an AGGREGATION node; this node is placed in the right branch of the split operated by MULTICAST. It takes the table grouped at the granularity of month and essentially performs a ROLL UP operation in data warehousing terms, transitioning from a finer granularity to a coarser one (i.e., the granularity at the year level). The sum at the month level, TotalCG_PerMonth, is further summed here to obtain TotalCG_PerYear.
- **OrderByYear2:** This is a SORT node; at this point, as extensively discussed in the SQL discussion¹¹, the goal is to match the two tables with granularity at year and month via the year attribute to then calculate the row-wise ratios. What needs to be done is a JOIN, but the MERGE JOIN operator in Microsoft SSIS requires, as mentioned earlier, that the tables to be merged be sorted according to the same criteria¹². Therefore, this node sorts the table with granularity year by year.
- **MergeJoinOnYear:** This is a MERGE JOIN node; the two tables are ready to be merged. This node performs a simple INNER JOIN on the year, and the output columns are year, month, TotalCG_PerMonth, and TotalCG_PerYear. This node does not have the option to configure error output.
- **CalculatePercentage:** This is a DERIVED COLUMN node; this node creates a new column using the following expression:

$$(DT_DECIMAL,2)ROUND((DT_DECIMAL,15)TotalCG_PerMonth / TotalCG_PerYear * 100,2)$$

For each row (i.e., for each month), the ratio between the total crime_gravity calculated for that month and that of the corresponding year is calculated. The numerator is cast to a decimal type to obtain a floating-point result, and this result is multiplied by 100 to get a percentage value. The ROUND function rounds to the second decimal place, and (DT_DECIMAL,2) removes insignificant zeros. The final result is named TotalGravityPercentage, and the error output has been configured as usual by redirecting rows with issues to a text file formatted as csv.

- **OrderByMonth:** This is a SORT node; this node sorts the resulting table by month and projects onto the columns month and TotalGravityPercentage.
- **OutputOnTextFile:** This is the usual FLAT FILE DESTINATION node that writes the result to the text file OutputQuery3.txt. Figure 10 displays the resulting file opened with Microsoft Excel.

¹¹Please refer to Appendix A.3

¹²In theory, starting from the left table, a lookup could be performed on the right table on the year key; this is because the right table has a distinct row for each year value. However, this table is not present in the database, and it should be created with an SQL command, but it was intentionally avoided to solve the queries using SQL commands within nodes.

3 Multidimensional Analysis

In this section, we will proceed with the creation of the OLAP cube (or Multidimensional Cube) using Microsoft SSAS and its deployment on the server. Subsequently, three queries will be resolved using the SSIS language provided by Microsoft SSMS, and finally, two dashboards will be created using Microsoft Power BI.

3.1 Assignment 0: Creation and deployment of the Cube

The construction phase of the OLAP Cube is managed in Microsoft SSAS through wizards, making it quite standard. However, the definition of measures and especially hierarchies within the different dimensions of the cube is crucial. Regarding measures, based on the business questions to be addressed, only the Crime Gravity measure has been created from the Custody table. As for dimensions, those related to Gun and Participant, not being useful for the business questions, were added only for completeness. Their hierarchies correspond to their primary keys, so they are flat hierarchies (without levels or members) and are therefore unnecessary. This means that these dimensions could have been omitted directly from the OLAP Cube. For the Geography and Date dimensions, additional hierarchies were created: GeoHierarchy and DayMonthQuarterYear, respectively. The GeoHierarchy consists of the members state, county, city, and geography_point¹³. DayMonthQuarterYear consists of the members year, quarter, month, and day. In Figure 12, we present the structure of the cube (with only the dimensions of interest expanded for space considerations).

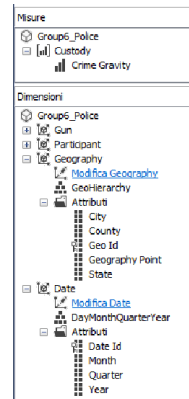


Figure 12: OLAP Cube

3.2 Assignment 1: SSIS Query 1

Query. *Show the total crime gravity for each city and the grand total with respect to the state.*

```
WITH MEMBER GrandTotalPerState AS
([Geography].[GeoHierarchy].CURRENTMEMBER.PARENT.PARENT, [Measures].[Crime Gravity])
SELECT {[Measures].[Crime Gravity], GrandTotalPerState} ON COLUMNS,
[Geography].[GeoHierarchy].[City] ON ROWS
FROM Group6_Police;
```

In this query, the multidimensional cube Group6.Police is selected to display the final result through the FROM clause. The SELECT clause declares which hierarchies will be placed on the columns and which on the rows. In particular, we want cities on the rows, and we retrieve them from the GeoHierarchy hierarchy. On the columns, we have placed the set composed of Crime Gravity (calculated at the granularity established by the member present on the rows, namely 'City') and a new measure to be created called GrandTotalPerState. The latter is created with the WITH MEMBER clause in the first part of the query; here, using the tuple operator (represented by parentheses), we calculate the Crime Gravity measure, i.e., the second term of the tuple, at the level established by the first element of the tuple, namely [Geography].[GeoHierarchy].CURRENTMEMBER.PARENT.PARENT. This level is nothing but the state to which the row element belongs; in fact, from the GeoHierarchy hierarchy, the current member¹⁴ is accessed through the CURRENTMEMBER operator, and from here, the hierarchies are traversed vertically two levels up by applying the PARENT operator twice. A portion of the result is shown in Figure 13. Due to the fact that there are many cities in each state, only the rows related to the first cities of the state of Alabama are displayed¹⁵.

	Crime Gravity	GrandTotalPerState
Alabama - Autauga County - Autaugaville	2	3983
Alabama - Autauga County - Chicago	2	3983
Alabama - Autauga County - Lincoln	0	3983
Alabama - Autauga County - Marietta	2	3983
Alabama - Autauga County - Nashville-Davidson	6	3983
Alabama - Autauga County - Odell	2	3983
Alabama - Autauga County - Prattville	24	3983
Alabama - Baldwin County - Baltimore	2	3983
Alabama - Baldwin County - Bay Minette	13	3983
Alabama - Baldwin County - Boston	2	3983
Alabama - Baldwin County - City of Birmingham	2	3983
Alabama - Baldwin County - City of Yorkers	1	3983
Alabama - Baldwin County - Daphne	13	3983
Alabama - Baldwin County - Denver	0	3983
Alabama - Baldwin County - East Chicago	2	3983
Alabama - Baldwin County - Elberta	8	3983

Figure 13: Result of the first SSIS query

¹³This column was created internally in Microsoft SSAS using the expression:

TRIM(city) + ' - ' + CAST(latitude AS VARCHAR) + ', ' + CAST(longitude AS VARCHAR).

¹⁴Here, the current member is the city, as established by what is reported in the ON ROWS clause of the SELECT statement

¹⁵This is caused by the default alphabetical sorting; therefore, since each city is identified by concatenating the terms related to its state of belonging, county, and the city itself, the first rows are all related to the same state (as well as the same county).

3.3 Assignment 2: SSIS Query 2

Query. Show the percentage increase or decrease in total crime gravity answers with respect to the previous year for each age group.

```
WITH MEMBER CG_YearlyVariation AS
IIF(ISEMPTY((PARALLELPERIOD(
[Date].[DayMonthQuarterYear].[Year],1,[Date].[DayMonthQuarterYear].CURRENTMEMBER),
[Measures].[Crime Gravity])), 'NO DATA AVAILABLE',
([Measures].[Crime Gravity] - (PARALLELPERIOD(
[Date].[DayMonthQuarterYear].[Year],1,[Date].[DayMonthQuarterYear].CURRENTMEMBER),
[Measures].[Crime Gravity])) / (PARALLELPERIOD([Date].[DayMonthQuarterYear].[Year],1,
[Date].[DayMonthQuarterYear].CURRENTMEMBER), [Measures].[Crime Gravity])),
FORMAT_STRING = 'PERCENT'
SELECT {[Measures].[Crime Gravity], CG_YearlyVariation} ON COLUMNS,
CROSSJOIN([Participant].[Age Group].[Age Group],
[Date].[DayMonthQuarterYear].[Year]) ON ROWS
FROM Group6_Police;
```

Messages	Results	Crime Gravity	CG_YearlyVariation
Adult 18+	2013	225	NO DATA AVAILABLE
Adult 18+	2014	7491	3229.33%
Adult 18+	2015	30027	300.84%
Adult 18+	2016	44478	48.13%
Adult 18+	2017	67820	52.48%
Adult 18+	2018	14635	-78.42%
Child 0-11	2013	0	NO DATA AVAILABLE
Child 0-11	2014	192	inf
Child 0-11	2015	726	278.13%
Child 0-11	2016	528	-27.27%
Child 0-11	2017	690	30.68%
Child 0-11	2018	186	-73.04%
Teen 12-17	2013	30	NO DATA AVAILABLE
Teen 12-17	2014	2412	7940.00%
Teen 12-17	2015	7929	228.73%

Figure 14: Result of the second SSIS query

year is not calculable due to a lack of data. A portion of the result is shown in Figure 14.

In this query, after accessing the cube, the Cartesian product between the Age Group and Year members of the two different hierarchies, Age Group of Participant and DayMonthQuarterYear of Date, is placed on the rows using the CROSSJOIN operator. On the columns, the set is formed by Crime Gravity calculated at the granularity established by the CROSSJOIN, and the new member CG_YearlyVariation is calculated as follows: The Crime Gravity of the previous year is calculated by navigating the DayMonthQuarterYear hierarchy using the PARALLELPERIOD operator. This quantity is subtracted from the Crime Gravity calculated at the granularity of the rows, and the result is divided by the Crime Gravity calculated again for the previous year, always using the PARALLELPERIOD operator, in order to obtain a percentage variation (appropriately formatted with the FORMAT_STRING clause). Finally, the IIF and ISEMPTY operators take into account the fact that for the first available year (in our case, 2013), the variation from the previous

3.4 Assignment 3: SSIS Query 3

Query. Show the ratio between the total crime gravity of each year w.r.t the previous year.

```
WITH MEMBER RatioWRT_PreviousYear AS
IIF(ISEMPTY((PARALLELPERIOD(
[Date].[DayMonthQuarterYear].[Year],1,[Date].[DayMonthQuarterYear].CURRENTMEMBER),
[Measures].[Crime Gravity])), 'NO DATA AVAILABLE',
FORMAT([Measures].[Crime Gravity] / (PARALLELPERIOD(
[Date].[DayMonthQuarterYear].[Year],1,[Date].[DayMonthQuarterYear].CURRENTMEMBER),
[Measures].[Crime Gravity]), '#0.000'))
SELECT {[Measures].[Crime Gravity], RatioWRT_PreviousYear} ON COLUMNS,
[Date].[DayMonthQuarterYear].[Year] ON ROWS
FROM Group6_Police;
```

In this query, access is made to the usual Group6_Police cube through the FROM clause. Starting from the DayMonthQuarterYear hierarchy, years are arranged on the rows in the SELECT clause. Within the same SELECT clause, a set is inserted consisting of two columns: the first is the Crime Gravity measure, and the second is RatioWRT_PreviousYear.

The latter is calculated by the conditional operator IIF within the WITH MEMBER clause as the string 'NO DATA AVAILABLE' in case the PARALLELPERIOD does not find a preceding element to the year of the current member (and this check is performed by the ISEMPTY operator). In the case where ISEMPTY returns FALSE, it is equal to the ratio of Crime Gravity for the current row divided by the Crime Gravity of the year preceding the current row. Naturally, even in this second case, navigation of the hierarchy is carried out through the PARALLELPERIOD operator. Figure 15 shows the result of the query (this time in its entirety since there are not many years in question).

Messages Results		
	Crime Gravity	RatioWRT_PreviousYear
2013	255	NO DATA AVAILABLE
2014	10095	39.588
2015	38682	3.832
2016	55113	1.425
2017	84872	1.540
2018	18679	0.220

Figure 15: Result of the third SSIS query

3.5 Assignment 4: Power BI Dashboard 1

Query. Create a dashboard that shows the geographical distribution of the total crime gravity in each age group.

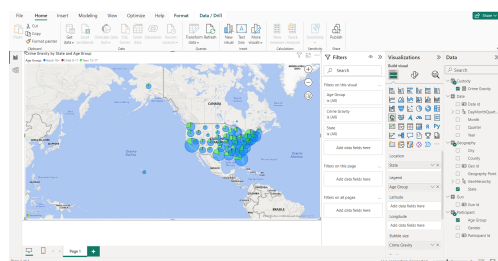


Figure 16: Dashboard of Assignment 4 Part 3.

groups.

In Figure 16, a dashboard created with Microsoft Power BI is depicted, showcasing the geographical distribution of total Crime Gravity within each age group. To create it, data was loaded from the server where the multidimensional cube had been deployed. In the Visualizations section, the Map icon was selected. In the Data section on the right side of the interface, the Crime Gravity measure was chosen, and the State member of the GeoHierarchy hierarchy was placed under the 'Location' field, allowing the software to associate each row with a specific geographic location (i.e., a particular state). This way, a bubble proportional to the Crime Gravity value was displayed for each state. At this point, the Age Group member was added to the 'Legend' field to ensure that the bubbles for each state depicted the distribution of the measure among the three different age

3.6 Assignment 5: Power BI Dashboard 2

In Figure 17, a dashboard is illustrated displaying the distribution of Crime Gravity restricted to the counties of the state of California, quarter by quarter, ordered by increasing Crime Gravity from top to bottom.

The cell coloring immediately highlights some important information:

Firstly, it is noticeable how the dataset is characterized by a generally increasing Crime Gravity from the earlier quarters to the more recent ones. This is likely not due to the fact that the more recent periods had the highest rate of serious crimes but rather because of the growing abundance of data over time.¹⁶

For this reason, in Figure 17, it was decided to present the lower right part of the dashboard, namely the one related to the most recent quarters (right side) of the most dangerous counties (bottom part).

Secondly, once our analysis is narrowed down to this portion of the dashboard, thanks to the more intense coloring of cells with higher Crime Gravity, it is easily evident, for example, that San Joaquin County faced high crime problems in the last quarter of 2017, as its Crime Gravity surged during that period (then stabilizing to a moderate value in the subsequent quarter). The same applies to Sacramento County during the second and third quarters of 2017, or Ventura County during the same period.

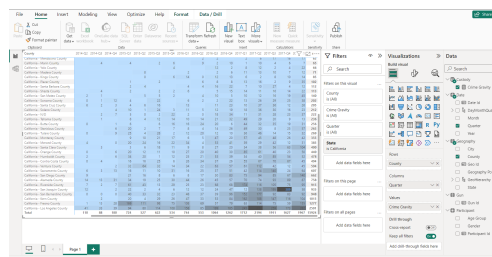


Figure 17: Dashboard of Assignment 5 Part 3.

¹⁶This can also be inferred by visualizing the variation over time in total Crime Gravity divided between males and females. It will be observed that the ratio between the measure calculated for males and that calculated for females is significantly higher in the earlier periods. This is not because the severity of crimes committed by women has suddenly grown exponentially, but simply because the data available for more recent periods is more complete.

Appendix A: SQL Solutions of Queries of Section 2

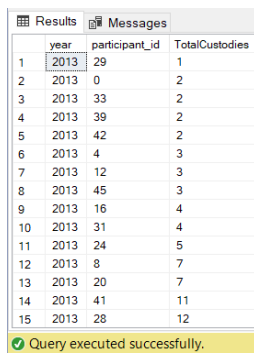
We briefly present the solutions to the queries from Section 2 in SQL language below.

A.1: Assignment 0 Part 2

Query. *For every year, list the participants ordered by the total number of custodies.*

A possible solution in SQL language is proposed below.

```
SELECT
    d.year,
    c.participant_id,
    COUNT(*) AS TotalCustodies
FROM
    Custody c
    INNER JOIN Date d ON c.date_id = d.date_id
GROUP BY
    d.year, c.participant_id
ORDER BY
    d.year, TotalCustodies, c.participant_id;
```



	year	participant_id	TotalCustodies
1	2013	29	1
2	2013	0	2
3	2013	33	2
4	2013	39	2
5	2013	42	2
6	2013	4	3
7	2013	12	3
8	2013	45	3
9	2013	16	4
10	2013	31	4
11	2013	24	5
12	2013	8	7
13	2013	20	7
14	2013	41	11
15	2013	28	12

Figure 18: Result of the first query

Here, starting from the Custody Table, an INNER JOIN is made with the Date Table. Grouping is done by year and participant identifier to have one row for each participant with the associated year to which the custody of that participant refers. At this point, a projection is made on the columns of interest, namely year and participant identifier, and for each formed group, the rows belonging to each group are counted; the new column representing this count has been renamed TotalCustodies.

Finally, the resulting table formed by the three columns year, participant identifier, and total number of custodies is sorted first by year (so that the first rows refer to the early years and the last rows to the recent years), then according to the total number of participants (so that in each group of the same year, rows with fewer total custodies appear first), and finally, since it is quite common for two participants to have the same total number of custodies in a given year, sorting is also done by participant identifier to improve the table's readability. The result of the query is shown in Figure 18.

A.2: Assignment 1 of Part 2

Query. *For each state, compute the stolen gravity index defined as the ratio between the total gravity of custodies involving stolen guns divided by the overall gravity of custodies.*

Here we present two different approaches to solving this query in SQL, followed by the solution using Microsoft SSIS. The first method creates two temporary tables (which could be two views) with two queries and calls these tables in a third query to provide the final result; it is more intuitive but less efficient. The second method, on the other hand, manages to solve the query "in one go," using a single SELECT, using some operators of the procedural language offered by Microsoft SQL Server¹⁷. The logic of this approach will be replicated to solve the query in Microsoft SSIS, thanks to the fact that it also offers the possibility of implementing conditional constructs.

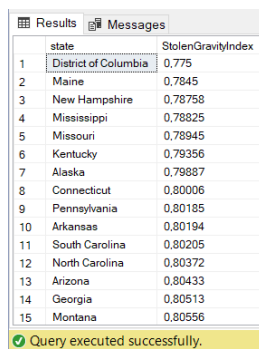
Conceptually, the logic with which the query is executed in the first formulation is very simple. A first SELECT block is executed to calculate the total Crime Gravity for each state, and the resulting table from this block is named TotalCG_Per.State; then, through a second SELECT block, the same thing is done but filtering for rows that have 'Stolen' in the is_stolen attribute, and the result of this SELECT block is named TotalStolenGravity; finally, the tables created in this way (which have the same number of rows, one for each

¹⁷The procedural language used to write procedures, functions, triggers, and other programming objects in the Microsoft SQL Server environment is called Transact-SQL (T-SQL). Transact-SQL is an extension of the standard SQL language with the addition of procedural constructs that allow the creation of logically complex data management operations.

state) are called in the two previous blocks, matched with a simple INNER JOIN on the state attribute, and starting from the two columns TotalCG_Per_State and TotalStolenGravity, the StolenGravityIndex ratio is calculated. The result is finally sorted in descending order with respect to the StolenGravityIndex measure.

Below is the SQL code in which the WITH clause is typical of T-SQL and is part of Common Table Expressions (CTEs), temporary queries defined within a SQL SELECT, INSERT, UPDATE, or DELETE statement. The WITH clause is used to declare a CTE and can be followed by one or more CTE definitions separated by commas.

```
WITH TotalCG_Per_State AS (  
    SELECT  
        ge.state,  
        sum(c.crime_gravity) AS TotalCG  
    FROM  
        Geography ge,  
        Custody c,  
        Gun gu  
    WHERE  
        ge.geo_id = c.geo_id AND  
        gu.gun_id = c.gun_id  
    GROUP BY  
        ge.state  
) ,  
  
TotalStolenGravity AS (  
    SELECT  
        ge.state,  
        sum(c.crime_gravity) AS TotalCG  
    FROM  
        Geography ge,  
        Custody c,  
        Gun gu  
    WHERE  
        ge.geo_id = c.geo_id AND  
        gu.gun_id = c.gun_id AND  
        gu.is_stolen = 'Stolen'  
    GROUP BY  
        ge.state  
)  
  
SELECT  
    t1.state,  
    ROUND(CAST(t2.TotalCG AS FLOAT) / t1.TotalCG, 2) AS StolenGravityIndex  
FROM  
    TotalCG_Per_State t1,  
    TotalStolenGravity t2  
WHERE  
    t1.state = t2.state  
ORDER BY  
    StolenGravityIndex;
```

	state	StolenGravityIndex
1	District of Columbia	0.775
2	Maine	0.7845
3	New Hampshire	0.78758
4	Mississippi	0.78825
5	Missouri	0.78945
6	Kentucky	0.79356
7	Alaska	0.79887
8	Connecticut	0.80006
9	Pennsylvania	0.80185
10	Arkansas	0.80194
11	South Carolina	0.80205
12	North Carolina	0.80372
13	Arizona	0.80433
14	Georgia	0.80513
15	Montana	0.80556

Figure 19: Result of the second Query

In the second approach, to solve the SQL query, a single SELECT clause is used. Starting from the Custody table, an INNER JOIN is performed with the Geography and Gun tables. The result is grouped by state, and for each group, the sum of the crime_gravity attribute values for rows with is_stolen = 'Stolen' is calculated. This is made possible by the application of the T-SQL CASE WHEN THEN ELSE END construct. Simultaneously (for the same groups), the sum of the crime_gravity attribute values for all rows in each group is also calculated. The ratio of these two calculated sums is then computed. Since these sums are performed on integer numbers, a float is obtained by casting either the numerator or denominator. In the proposed query, the numerator is cast to float within the CASE WHEN THEN ELSE END construct. The entire ratio is rounded using the ROUND construct and renamed as StolenGravityIndex. The only projected attributes are state and the StolenGravityIndex ratio. The code for the query just described is shown below, and the result of the query executed on the datamart on Microsoft SQL Server is shown in Figure 19.

```
SELECT
    ge.state,
    ROUND(SUM(CASE WHEN gu.is_stolen = 'Stolen' THEN CAST(c.crime_gravity AS FLOAT) ELSE 0 END)
        / SUM(c.crime_gravity), 5)
    AS StolenGravityIndex
FROM
    Custody c
    INNER JOIN Geography ge ON c.geo_id = ge.geo_id
    INNER JOIN Gun gu ON c.gun_id = gu.gun_id
GROUP BY
    ge.state
ORDER BY
    StolenGravityIndex;
```

A.3: Assignment 2 Part 2

Query. For each month, compute the total gravity in percentage with respect to the annual total.

For this query, we will propose three solutions using different SQL approaches, and then demonstrate how to solve the problem using Microsoft SSIS. The first solution employs temporary tables and the WITH clause; as mentioned earlier, this solution is intuitive but less efficient. The second solution utilizes the OVER PARTITION BY clause, an extended SQL clause. The third solution involves two nested queries, serving as inspiration for setting up the logic for solving the query in Microsoft SSIS.

Now, let's proceed with the first solution using the WITH clause and temporary tables.

```
WITH TotalGravityMonth AS (
    SELECT
        d.year,
        d.month,
        SUM(c.crime_gravity) AS TotCG_per_month
    FROM
        Custody c,
        Date d
    WHERE
        c.date_id = d.date_id
    GROUP BY
        d.year,
        d.month
),

TotalGravityYear AS (
    SELECT
```

```

        d.year,
        SUM(c.crime_gravity) AS TotCG_per_year
FROM
    Custody c,
    Date d
WHERE
    c.date_id = d.date_id
GROUP BY
    d.year
)

SELECT
    TGM.month,
    ROUND(CAST(TGM.TotCG_per_month AS FLOAT) / TGY.TotCG_per_year * 100, 2) AS
        TotalGravityPercentage
FROM
    TotalGravityMonth TGM,
    TotalGravityYear TGY
WHERE
    TGM.year = TGY.year;

```

Here, the idea is to create a table where, for each month, you have the sum of crime.gravity, and another table where this sum is provided for each year. Afterward, through a third SELECT statement, these two tables are called and correlated using the 'year' attribute. The required ratio is then calculated. Specifically, the first temporary query performs an INNER JOIN between Custody and Date. It groups by year and month, and for each created group, calculates the sum of crime.gravity values of the rows belonging to each group. The resulting table is renamed as TotalGravityMonth. The second query does the same but limits the grouping to the granularity of the 'year' attribute. The resulting table is renamed as TotalGravityYear. The final SELECT statement performs an INNER JOIN between TotalGravityMonth and TotalGravityYear on the 'year' attribute. At this point, there is a table with one row for each month, along with the respective reference year, the total sum of crime.gravity for that month, and the total sum of crime.gravity for that year. The next step is to calculate the row-wise ratio of the TotalGravityMonth and TotalGravityYear attributes to obtain the desired percentage. Finally, projecting onto this percentage and the reference month. Below is another alternative resolution method, very similar to the previous one, where the query is resolved without using temporary tables.

```

SELECT
    d.month,
    CAST(
        ROUND(SUM(c.crime_gravity) * 100.0 / (
            SELECT
                SUM(c2.crime_gravity)
            FROM
                Custody c2
                JOIN Date d2 ON c2.date_id = d2.date_id
            WHERE
                d2.year = d.year
        ), 2
    ) AS DECIMAL(10,2)) AS TotalGravityPercentage
FROM
    Custody c
    JOIN Date d ON c.date_id = d.date_id
GROUP BY
    d.year,
    d.month;

```

The logic is exactly the same as the previous method, and we won't spend much time here discussing the details. However, it should be emphasized that, although this solution is less readable, it does not involve unnecessary resource consumption like generating temporary tables. Instead, it allows the DBMS to optimize the query with various mechanisms typical of the physical implementation of the database. This solution is

presented because it shares a logic very similar to what will be used later to solve the query in Microsoft SSIS. The code for this second method is provided below.

Below is a third method that can be used to solve this query, employing a combination of two operators from the extended SQL language, namely OVER and PARTITION BY, which are particularly well-suited for resolving queries of this type. Indeed, the combination of these operators allows the creation of the operation that was previously attempted manually, involving two different types of grouping (on two different granularities) while keeping them on the same table. The code is provided below.

```
WITH MonthlyGravity AS (
    SELECT
        d.year,
        d.month,
        SUM(c.crime_gravity) AS MonthlyTotalGravity,
        SUM(SUM(c.crime_gravity)) OVER (PARTITION BY d.year) AS AnnualTotalGravity
    FROM
        Custody c
    JOIN Date d ON c.date_id = d.date_id
    GROUP BY
        d.year,
        d.month
)

SELECT
    month,
    CAST(
        ROUND(MonthlyTotalGravity * 100.0 / AnnualTotalGravity, 2) AS DECIMAL(10,2)
    ) AS TotalGravityPercentage
FROM
    MonthlyGravity;
```

In the first SELECT, an INNER JOIN is performed between the Custody table and the Date table. Rows are grouped at the granularity of month, and two aggregations are carried out:

- The first aggregation is the standard one that sums the values of the crime_gravity measure at the granularity level for which we have grouped, i.e., at the month level.
- The second aggregation calculates the cumulative sum of crime_gravity for each year. The function SUM(c.crime_gravity) calculates the total sum of crime gravity for each row resulting from the main query, and the OVER (PARTITION BY d.year) operator ensures that this cumulative sum is done separately for each year. The result is labeled as AnnualTotalGravity.

From the table resulting from this SELECT, a row-wise ratio is calculated between the sums at the month granularity and those at the year granularity through a second SELECT block. The rounding and casting operations are the same as discussed earlier and are performed to achieve the result equal to that obtained by the other proposed methods. This result is shown in Figure 20.

Results		Messages
	month	TotalGravityPercentage
1	2013-01	6.67
2	2013-02	8.63
3	2013-03	14.90
4	2013-04	9.02
5	2013-05	11.76
6	2013-06	9.80
7	2013-07	7.06
8	2013-08	7.84
9	2013-09	11.37
10	2013-10	5.49
11	2013-11	3.92
12	2013-12	3.53
13	2014-01	6.48
14	2014-02	4.68
15	2014-03	7.08

Query executed successfully.

Figure 20: Result of the third query