UNIVERSITÀ DI PISA

Department of Computer Science

Master programme in Data Science and Business Informatics

Decision Support Systems
Module II: Laboratory of Data Science
Project report

Miccoli Martin
Montinaro Aldo
Poiani Marco

A.A. 2024/2025

Contents

Introduction

Part 1

The goal of the assignments of Part 1 is to design and populate a database using data from given CSV files and perform a series of tasks to manipulate and analyze this data. We are required to use python scripts and additionally implement solutions using SQL Server Integration Services (SSIS), with a focus on minimizing the use of SQL commands within the nodes, relying primarily on native SSIS nodes for computations.
This project revolves around traffic incidents in Chicago, using a simplified dataset derived from Kaggle [2]. The goal is to simulate a Decision Support System tailored for an insurance company. The dataset consists of three main files:

1. **Crashes.csv**: contains detailed records of traffic incidents occurring between January 2014 and January 2019 in Chicago, with additional information about the incidents' causes, road properties, and injuries reported.
2. **People.csv**: provides details about individuals involved in the traffic incidents, including their sex, age, and city of residence.
3. **Vehicles.csv**: contains information about the vehicles involved in the incidents, including data collected by the police post-incident.

Part 2

The second part of the project requires answering business questions using a datacube created (using SSAS) on the basis of the database prepared in the first part of the work. Some business questions will be then answered using the MDX language (in SQL Server Management Studio). The final part of the project involves the creation of three dashboards containing useful insights related to the insurance context regarding the incidents analyzed.

Project Assignment - Part 1

2.1 Data Understanding and Cleaning

Assignment 1-2

The **Crashes** dataset contains 257925 records with 36 attributes with information about road incidents in Chicago between January 18, 2014 and January 12, 2019. The data types found are int, float and object (mainly strings but also datetimes formatted as strings). There are no duplicate rows, but there are missing values that have been handled using different approaches:

- REPORT_TYPE: binary column which contains 1.94% of nulls; cannot be retrieved and we decided to keep null values in order to not assign incorrect labels.
- STREET_DIRECTION: contains cardinal directions, only 2 missing values that cannot be retrieved.
- BEAT_OF_OCCURRENCE: 4 nulls that can be retrieved by spatial join with official police beat partition from Chicago Data Portal [1].
- MOST_SEVERE_INJURY: 7 nulls, not available online.
- LATITUDE, LONGITUDE, LOCATION: first two contains 0.4% of nulls, while LOCATION contains 1022 nulls. All of them can be retrieved using reverse geocoding.

A candidate key for the records is the column **RD_NO** which contains 257925 unique values (no missing) with equal frequency 1 and constant length 8. We can get some initial insights from Crashes: by looking at distributions of days and hours of accidents we see that the most of them occurred on a Friday, during the month of Ocrober and around 3/4 PM; in 79.64% of cases weather is CLEAR and most of the accidents happened on DRY roadway surface; the LOCATION with the highest number of records corresponds to a place near Chicago-O'Hare International Airport.

The Crashes data cleaning was handled by `crashesDataCleaning.py` script that orchestrates the cleaning workflow calling objects (and corresponding functions) from `utils_crashes.py` script. The libraries used in this step are: `csv`, `time`, `geopy`, `shapely`, `h3`. The classes containing cleaning functions are:

- `DataProcessor`: contains loading and saving functions, dates processing (we separated CRASH_DATE into day, month, quarter, year, hour and minute) and the correction of NUM_UNITS values (based on number of UNIT_NO in the corresponding data in Vehicles.csv);
- `SpatialOperator`: by geocoding and reverse geocoding retrieves geospatial information, performs spatial joins to retrieve police beats and gets H3 encoding to enlarge spatial information; indentifies recorded location that falls outside the geographical boundaries of the city of Chicago and correct them using reverse geocoding.

Main script (file `crashesDataCleaning.py`) takes Crashes, Vehicles, PoliceBeats and generates a clean and updated version of Crashes.

The **People** dataset contains 564565 records and 19 variables that provide both demographic information about individuals involved in crashes and details about their condition at the time of the crash. Initially, the variable types were float for VEHICLE_ID, AGE and DAMAGE. However, after preprocessing, VEHICLE_ID and AGE were converted to int, which is more appropriate for these types of data. All other variables were of type object, mainly strings, while CRASH_DATE was formatted as datetime. The only feature removed from the dataset was CRASH_DATE, as the same information is available in the Crashes dataset. The data in People contains some inconsistencies (e.g., drivers with an age < 16). However, for the scope of this project, we addressed only those inconsistencies that could lead to errors during table creation. For example, there were no missing values for PERSON_ID, no duplicate rows, and, for the DAMAGE column, which was crucial for the fact table, we addressed missing values (74309 in total), occurred only in cases where the DAMAGE_CATEGORY was \$0–500. In such cases, we assigned the maximum value, \$500, to account for the worst-case scenario. For the foreign keys, in the dataset, there are RD_NO which has no missing values and VEHICLE_ID which presents some missing values. By exploiting RD_NO the dataset can be joined with the dataset about crashes, while by using VEHICLE_ID, where not missing, and PERSON_ID can be merged with the dataset about vehicles. PERSON_ID is the primary key of the table and presents a prefix consisting of a letter, "P" if the person was a passenger, and "O" in all the other cases. In the whole dataset exist 6 exceptions where the PERSON_ID doesn't appear in the vehicle dataset and their VEHICLE_ID was missing, in these cases, there was no way to link the person with a vehicle. The last inconsistency that we handled involved the variable CITY, sometimes instead of

reporting the name of the city the value was the ZIP code, so we retrieved the city's name from it. Another issue was the name of the same city written with typos, we attempted to manage them but then we left them as they were because, for the project, they didn't cause any trouble.

The **Vehicles** dataset contains 357234 records with detailed information about vehicles involved in reported crashes. The data cleaning procedure on this dataset was managed using `vehiclesDataCleaning.py`: a custom DataFrame class was developed to validate and clean the vehicle data. This included checking the consistency of each row's column count and cleaning extraneous details from the MODEL column to make vehicle names consistent. Missing VEHICLE_ID values were filled with a default of -1 to ensure consistency. To enhance clarity we also modified the MODEL column which initially contained excessive text in most rows (we removed everything inside parenthesis or after a comma).

2.2 Data Warehouse Schema

Assignment 3

We identify as a fact table a table called **DamageToUser** that will contain, in addition to the foreign keys for the dimensions, the measures DAMAGE (containing the estimated cost of the damage) and NUM_UNITS (containing the number of units involved in the incident associated with the user under consideration); each fact (identified by DTUID) will represent a new damage associated to a user involved in an accident. We came to this choice also because we noticed, in the People dataset, the presence of totally identical rows (thus same person with the same characteristics) in which only the value of the associated DAMAGE changed. The primary analytical focus will be on understanding damage outcomes for individuals involved in crashes. We defined a **star schema** (to better support efficient query performance) containing 8 dimensional tables:

- `DateDimension`: breaks down the crash date into granular components like year, quarter, hour and so on.
- `PersonDimension`: contains details about the individual involved in a crash. We assume that data was taken so that PERSON_ID with each new crash will have a new value even if the individual is the same; maybe we can assume that there exists (perhaps created to respect privacy) a table accessible only to police departments in which each uniquely identified individual is associated with a list of one or multiple PERSON_ID that appear in our data. For the purposes of our study we will however consider PERSON_ID as the unique ID for the PersonDimension table. This table contains also INJURY_CLASSIFICATION to understand the type of injury associated with each person.
- `VehicleDimenion`: contains information about vehicles involved in reported crashes, including make, model and type. As key we use VEHICLE_ID because for people without a corresponding vehicle it does not make sense to keep data in this table because the resulting row would be empty except for CRASH_UNIT_ID.
- `CrashReportDimension`: includes report details identified by RD_NO.
- `CauseDimension`: isolates contributory causes and driver-related factors.
- `InjuryDimension`: contains injury metrics, which corresponds to the number of individuals with each specific type of injury recorded.
- `LocationDimension`: includes geospatial data and information about traffic control devices and road conditions.
- `WeatherDimension`: contains information about weather and lighting.

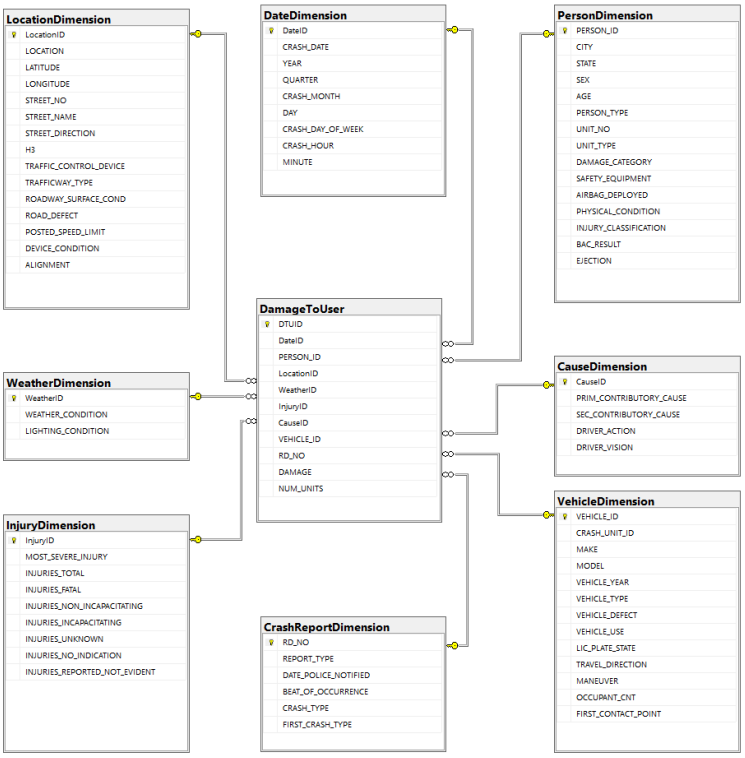


Figure 2.1: Data warehouse schema

Analyzing the chosen schema we saw that the PersonDimension table could contain Slowly Changing Dimensions: we could handle CITY and STATE as type 2 SCDs and AGE as type 1. As described earlier, however, if the same individual is re-entered into the database because he or she is involved in a new incident it will be associated with a new PERSON_ID, so in our case it would be counterproductive to manage these attributes.

2.3 Data Preparation and Uploading

Assignment 4

After completing the DW schema design, we developed a script capable of splitting the data from the three original datasets into the different tables in the diagram. First, we defined a dictionary containing the designed schema (table names as keys, lists of feature names as value for each key); we used the same attribute names of the original datasets but we also added new ID columns for some tables (DateID, LocationID, WeatherID, InjuryID, CauseID). We inserted some **preprocessing functions** to normalize and clean IDs that convert them into integer and into uppercase if they contain letters removing also spaces if present; this procedure solved some mismatches we encountered during first tests of merging the data. To remove further redundancy we also created a function that **filter out duplicate entries** after merging. The procedure continues with an indexing function that prepares the original data to be merged in a single dataset: the merging process is guided by RD_NO and VEHICLE_ID and contains debug statements to output missing or incorrect rows.
Finally, the **split function** splits the merged_data into the tables of the DW schema assigning a unique ID to each table. VehicleDimension invalid vehicle ID are managed differently to avoid inaccurate data: a specific function removes NaN vehicles from the final dimension table to avoid useless data. The fact table is created and populated in the last step to ensure correct handling of foreign keys.

Assignment 5

The main steps of our upload script are: infer the data type for each feature from a sample row of each table, infer the table schema itself from the header, create the table on the database, and load the data after validating it with a control function.

Assignment 6

We first created a python script that duplicates the existing tables in the database keeping the schema and constraints. To handle the upload of 10% of the data on the duplicate table set (_SSIS) via Integration Service, we started by randomly populating the fact table DamageToUser_SSIS with 10% of the data via the Percentage Sampling node. After that we apply the same data flow for each table, inserting them one after another within the same Control Flow as the fact table upload; the nodes for each table are: OLE DB Source (original table), first Lookup to look for the match with the foreign key in the fact table, second Lookup to check that the record is not already in the new table, OLE DB destination for the _SSIS version table.

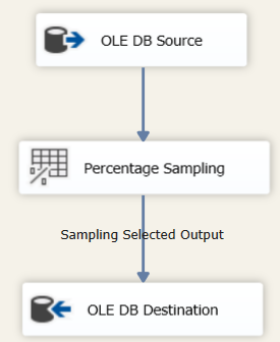


Figure 2.2: Data Flow task that samples 10% of the data from original table and upload them on the DamageToUser_SSIS table

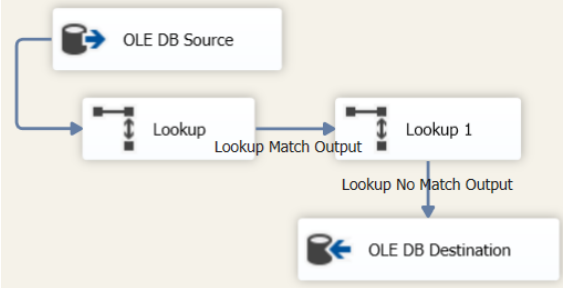


Figure 2.3: Data Flow task (same for all the tables) that populates the dimension tables

We also implemented in the same solution a second package with the same goal, but instead of retrieving the data for sampling from the original tables, it gets it from the files from the split of the original datasets. The only difference from the data flow logic seen above is that we need to add a Derived Column node to convert via type casting the columns that need to match with the data types in the OLE DB destination.

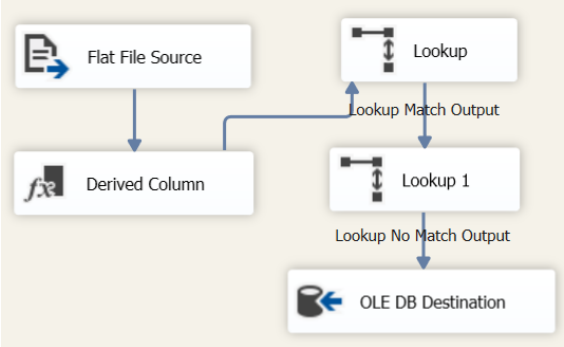


Figure 2.4: Data Flow task that populates dimension tables using data coming from CSV split files (same flow for all the dimension tables)

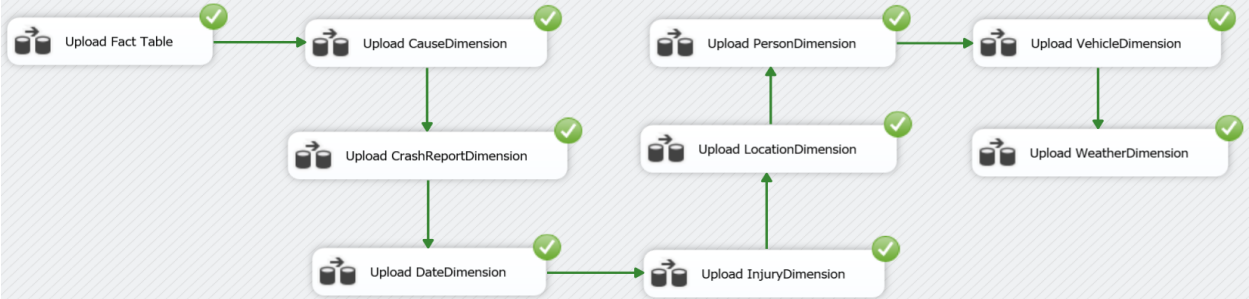


Figure 2.5: Full Control Flow of the sampling package

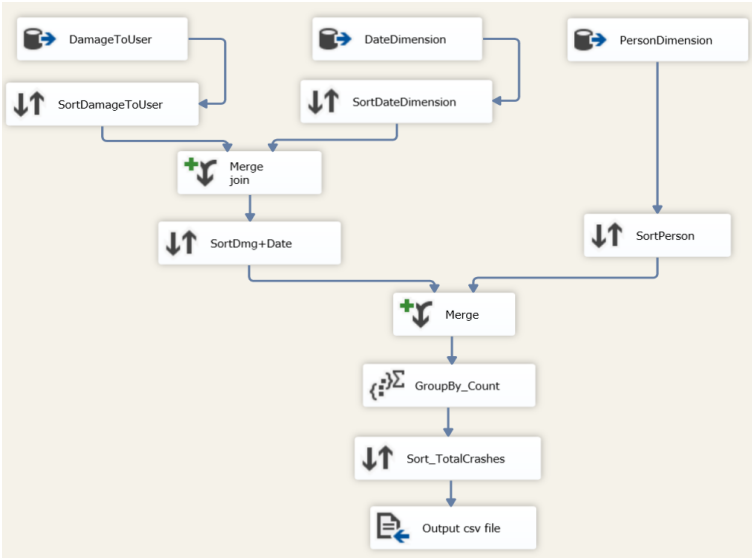
2.4 SSIS Business Questions

All the following business questions are included in the same Control Flow panel of a package called `Assignments_6a7a8a9a`.

Assignment 6a

For every year, show all participants ordered by the total number of crashes

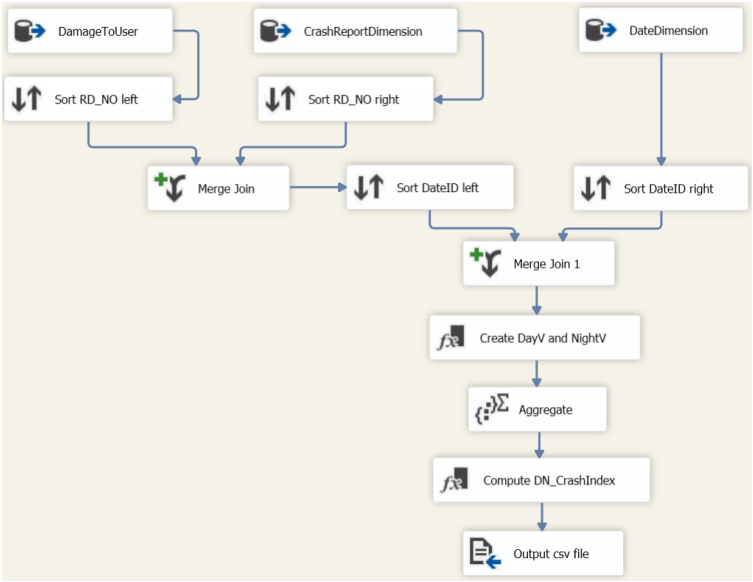
We implemented a single Data Flow Task in the Control Flow panel with three OLE DB sources: DamageToUser and DateDimension (merged after sorting by DateID) and PersonDimension (merged to the previous after sorting by PERSON_ID). On the merged dataset we applied an Aggregation transformation to group by PERSON_ID and YEAR, counting RD_NO (as "Total_Crashes"). We saved the sorted data by Total_Crashes in a Flat File Destination in csv format.



Assignment 7a

For every police beat, compute the day-night crash index, defined as the ratio between the number of vehicles (NUM_UNITS) involved in an incident between 9 pm and 8 am, and the number of vehicles involved in an incident between 8 am and 9 pm

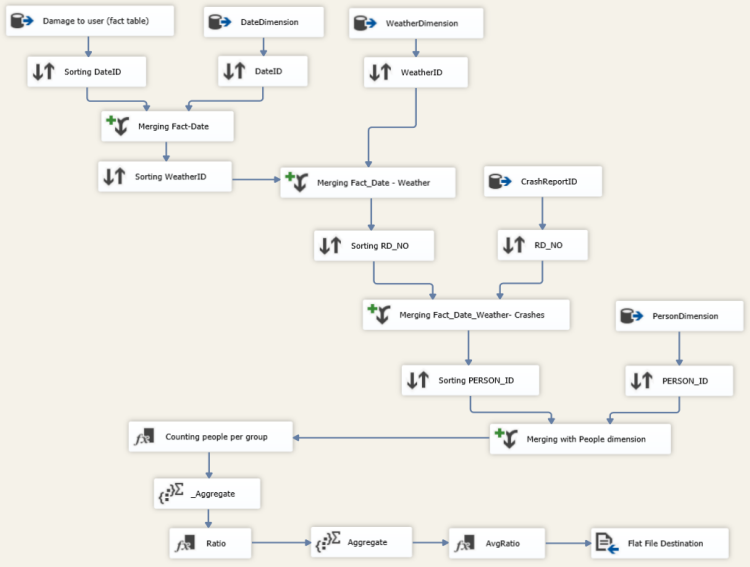
In this case we first did a join between DamageToUser and CrashReportDimension, and then did a merge again with DateDimension. To the resulting dataset, we add two Derived Columns (DayVehicles and NightVehicles) containing the NUM_UNITS of daytime crashes and those of nighttime crashes based on the time slots 8-21 and 21-8. An Aggregate transformation then groups by BEAT_OF_OCCURRENCE and sums the values of DayVehicles and NightVehicles. Before exporting the result to csv as Flat File Destination, a new Derived Column "DN_CrashIndex" is added, which contains the ratio of the number of nighttime accidents to the number of daytime accidents.



Assignment 8a

For each quarter, weather condition, and beat, show the average ratio of people under 21 years old to people over 21 years old involved in crashes

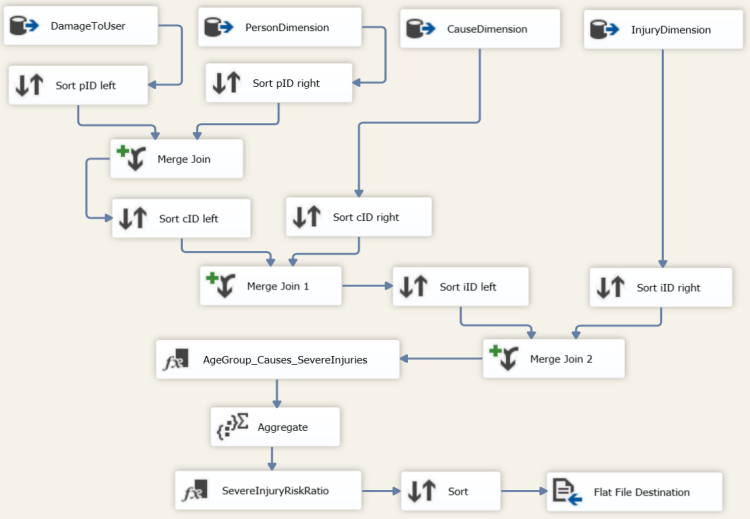
We used again single Data Flow Task with 5 OLE DB sources for retrieving data from DamageToUser, DateDimension, WeatherDimension, CrashReportDimension and PersonDimension selecting only the features required for the query and the result. We sorted the features, merged the data and added a Derived Column in which we indicated for each RD_NO if the person involved was under 21 or over 21. After that, we grouped by Quarter, Weather Condition, Beat of occurrence and RD_NO for counting for each RD_NO how many people were under 21 and how many were over 21. In the next step we inserted another Derived Column that computes the ratio (U21/O21) for each RD_NO and finally by grouping rows by Quarter, Weather condition and Beat of occurrence we computed the average ratio.



Assignment 9a

How do different age groups and contributory causes influence injury severity and average damage costs in traffic accidents?

After merge join between DamageToUser, PersonDimension, CauseDimension, and InjuryDimension, we created three derived columns: AgeGroup contains the age group (U21, 21-30, 31-50, 51-65, 66+), Causes contains a string with PRIM_CONTRIBUTORY_CAUSE and SEC_CONTRIBUTORY_CAUSE, and SevereInjuries as the sum of INJURIES_FATAL and INJURIES_INCAPACITATING. After an Aggregate in which we group by AgeGroup and Causes by summing SevereInjuries and INJURIES_TOTAL and averaging DAMAGE, we again create a Derived Column "SevereInjuryRiskRatio" as the ratio of SevereInjuries to INJURIES_TOTAL. We export the results by sorting decreasingly by the computed ratio and the average DAMAGE.



Project Assignment - Part 2

3.1 SSAS - Data Cube

Assignment 1

To create the cube necessary for performing MDX queries, we used SSAS. Initially, we added only the measures and dimensions required to answer the queries. However, our final decision was to include all dimensions that might be needed for future work. The first two measures defined were DAMAGE and NUM UNITS. During the development of the queries, we calculated additional measures to optimize query execution. These calculated measures will be discussed in the context of the specific queries in which they were used. Regarding dimensions, we defined the following: **Cause**, **Crash Report**, **Date**, **Injury**, **Location**, **Person**, and **Vehicle**, as they were essential for the queries. Furthermore, we added the **Weather** dimension, even though it was not utilized in any of the queries, with the intention of supporting potential future use cases. Additionally, we defined several hierarchies to structure the data effectively. The first hierarchy is for the Date dimension, which includes the levels: **Year**, **Quarter**, **Month**, **Day**, **Hour**, and **Minute**. Another hierarchy is defined for the Cause dimension, consisting of the levels: **Primary Contributory Cause** and **Secondary Contributory Cause**. Furthermore, we defined three additional hierarchies: StreetNameNo for Street dimension, with the levels **Street Name** and **Street Number**, StateCity for Person dimension, with the levels **State** and **City** and MakeModelYear for the Vehicle dimension, with the levels: **Make**, **Model**, and **Year**. All other dimensions are flat and do not contain any hierarchies.
Finally, we created a named calculation `AgeGroup` as a new Person Dimension column, containing the age groups chosen for Assignment 9a seen earlier in 2.4.

3.2 MDX Queries

The purpose of this section is to illustrate the OLAP querying techniques applied to analyze the data stored in the `[GROUP_8_CUBE]` cube using the MDX query language. Each query was designed to address specific business questions using custom calculations and multidimensional hierarchies.

Assignment 3

Compute the average yearly damage costs as follows: for each crash, calculate the total damage to the user divided by the number of distinct people involved in the crash. Then, compute the average of these values across all crashes in a year.

```
SELECT
[Measures].[TotalAvgDamageInCrashes] ON
    ↪ COLUMNS,
NONEMPTY([Date Dimension].[YEAR].[YEAR])
    ↪ ON ROWS
FROM [GROUP_8_CUBE]
```

	TotalAvgDamageInCrashes
2014	$3,233.44
2015	$2,365.50
2016	$2,445.12
2017	$2,546.08
2018	$2,591.91
2019	$2,700.35

We used calculated members to obtain the values of the number of people involved in each crash, the average damage per user and the total average damage in a set of crash reports. Since the calculated members used could also be useful for other types of analysis, we decided to insert them as precomputed members within the cube with SSAS; at this point among the cube measures we will have `AvgDamagePerUser`, `TotalAvgDamageInCrashes` and `nPeopleInvolved`. We also created a precomputed set `ValidCrashes` to filter crash reports with `nPeopleInvolved` and total damage greater than zero. Below are definitions of the named set and calculated members.

```
WITH
    MEMBER nPeopleInvolved AS DISTINCTCOUNT([Person Dimension].[PERSON ID].[PERSON ID])
    MEMBER AvgDamagePerUser AS [Measures].[DAMAGE] / [Measures].[nPeopleInvolved]
    SET ValidCrashes AS FILTER(
        [Crash Report Dimension].[RD NO].Members,
        [Measures].[nPeopleInvolved] > 0 AND [Measures].[DAMAGE] > 0)
    MEMBER TotalAvgDamageInCrashes AS AVG([ValidCrashes], [Measures].[AvgDamagePerUser])
```

Assignment 4

For each location, show the damage costs increase or decrease, in percentage, with respect to the previous year.

In this case, we work only on the MDX side without creating precomputed members on the SSAS side, because we use the PrevMember function that makes the member dynamic and evaluated at query time. First we go to identify the DAMAGE of the year before that in the current member (`DamagePreviousYear`), then we use it by calculating the difference with the DAMAGE of the current member, dividing by the damage of the previous year (`DamageChangePercentage`), so as to get a percentage value of the change. In our database each LOCATION is stored as a POINT object.

```
WITH
    MEMBER DamagePreviousYear AS
        ([Date Dimension].[YEAR].CURRENTMEMBER.
            ↪ PREVMEMBER, [Measures].[DAMAGE])
    MEMBER DamageChangePercentage AS
        IIF(ISEMPTY(DamagePreviousYear), NULL,
            ([Measures].[DAMAGE] -
                ↪ DamagePreviousYear) /
                ↪ DamagePreviousYear
        ), FORMAT_STRING = 'Percent'
SELECT
    NONEMPTY(([Date Dimension].[YEAR].[YEAR],
        ↪ DamageChangePercentage)) ON COLUMNS,
    NONEMPTY([Location Dimension].[LOCATION].[
        ↪ LOCATION]) ON ROWS
FROM [GROUP_8_CUBE];
```

	2015	2016	2017	2018	2019	Unknown
	DamageCh...	DamageCh...	DamageCh...	DamageCh...	DamageCh...	DamageCh...
POINT (-87.528378024491 41.695348560361)	(null)	(null)	(null)	(null)	-100.00%	(null)
POINT (-87.528407531546 41.706262650068)	(null)	(null)	3142.99%	-100.00%	(null)	(null)
POINT (-87.528526070387 41.706270368411)	(null)	(null)	(null)	-100.00%	(null)	(null)
POINT (-87.528900084421 41.707068262805)	(null)	(null)	(null)	806.64%	-100.00%	(null)
POINT (-87.52909613429 41.702385146499)	(null)	(null)	(null)	-12.83%	-100.00%	(null)
POINT (-87.529096549916 41.702328903582)	(null)	(null)	(null)	(null)	-100.00%	(null)
POINT (-87.529186180472 41.702810877224)	(null)	(null)	-100.00%	(null)	(null)	(null)

Assignment 5

For each quarter, show all the locations where the number of vehicles involved exceeds the average number of vehicles involved in the corresponding quarter of the previous year. Also, report the increase as percentage.

We used AVG and PARALLELPERIOD to compute the previous year's average DAMAGE dynamically by shifting the hierarchy back one year. To measure the percentage increase, we defined a calculated member with the IIF function, ensuring that the computation occurred only when the average was non-zero. The filtering isolates records where the number of vehicles is higher than the calculated average for the corresponding period. Rows in the output are organized by year, quarter, and location.

```
WITH MEMBER AvgVehiclesPrevYear AS
    AVG(PARALLELPERIOD(
        [Date Dimension].[DateHierarchy].[YEAR], 1,
        [Date Dimension].[DateHierarchy].CURRENTMEMBER), [
            ↪ Measures].[NUM UNITS] ), FORMAT_STRING = "#.##"
  MEMBER PercentageIncrease AS
    IIF(AvgVehiclesPrevYear > 0,
        ([Measures].[NUM UNITS] - AvgVehiclesPrevYear) /
            ↪ AvgVehiclesPrevYear,NULL),
    FORMAT_STRING = 'Percent'
SELECT {[Measures].[NUM UNITS], AvgVehiclesPrevYear,
    ↪ PercentageIncrease} ON COLUMNS,
FILTER(([Date Dimension].[YEAR].[YEAR],
    [Date Dimension].[QUARTER].[QUARTER],
    [Location Dimension].[LOCATION].[LOCATION]),
    [Measures].[NUM UNITS] > AvgVehiclesPrevYear AND
        ↪ AvgVehiclesPrevYear <> 0) ON ROWS
FROM [GROUP_8_CUBE]
```

			NUM UNITS	AvgVehiclesPrevYear	PercentageIncrease
2016	3	POINT (-87.56596963127589 41.73506895)	5	4.	25.00%
2016	3	POINT (-87.566649893195 41.768028469262)	8	1.	700.00%
2016	3	POINT (-87.575328550664 41.751503939851)	4	1.	300.00%
2016	3	POINT (-87.576082625984 41.758986864707)	6	4.	50.00%
2016	3	POINT (-87.576345488656 41.77336021958)	12	1.	1100.00%
2016	3	POINT (-87.57644168577 41.765974097973)	4	1.	300.00%
2016	3	POINT (-87.576443122779 41.766027779516)	36	4.	800.00%
2016	3	POINT (-87.576638552177 41.773368408547)	14	4.	250.00%

Assignment 6

For each vehicle type and each year, show the information and the (total) damage costs of the person with the highest reported damage.

As "information" we decided to show PERSON_ID, SEX, AGE, UNIT_NO (the last one in particular because it allows us to understand whether the identified user is responsible for the incident). To develop the query, we created a named set `TopPersonByDamage` using the `GENERATE` function, in which the parent set is a cartesian product of all members of the YEAR hierarchy from the Date Dimension and all members of the VEHICLE TYPE hierarchy from the Vehicle Dimension; the result is a set of all `(Year, Vehicle Type)` pairs. The subquery containing information extraction, filtering of nonempty values, and selection of the combination having the highest DAMAGE among all the filtered ones is applied to each pair. Then the result of GENERATE will be a single set containing the combinations `(Year, Vehicle Type, Top Person)`.

```
WITH SET TopPersonByDamage AS
    GENERATE(
      ([Date Dimension].[DateHierarchy].[YEAR],
        [Vehicle Dimension].[VEHICLE TYPE].[VEHICLE TYPE]),
      TOPCOUNT(
        NONEMPTY(
          ([Person Dimension].[PERSON ID].[PERSON ID],
          [Person Dimension].[SEX].[SEX],
          [Person Dimension].[AGE].[AGE],
          [Person Dimension].[UNIT NO].[UNIT NO]),
          [Measures].[DAMAGE]
        ),
        1,
        [Measures].[DAMAGE]
      )
    )
SELECT [Measures].[DAMAGE] ON COLUMNS,
  NONEMPTY(
    ([Date Dimension].[DateHierarchy].[YEAR],
    [Vehicle Dimension].[VEHICLE TYPE].[VEHICLE TYPE],
    TopPersonByDamage)
  ) ON ROWS
FROM [GROUP_8_CUBE]
```

						DAMAGE
2014	PASSENGER	O24496	F	41	2	8248.14
2014	UNKNOWN/NA	O12093	U	0	1	566.67
2015	ALL-TERRAIN VEHICLE (ATV)	O1247	M	17	1	3960.72
2015	BUS OVER 15 PASS.	O14336	M	70	1	10025.26
2015	BUS UP TO 15 PASS.	O443	F	30	2	9215.69
2015	MOTOR DRIVEN CYCLE	O13502	M	33	2	4318.66
2015	MOTORCYCLE (OVER 150CC)	O10548	M	44	2	6081.83

Assignment 8.2

For each year, show the most risky crash type and its total damage costs. To measure how risky a crash type is, you should assign a weight to each type of injury you encounter in the data (for example, a fatal injury weighs 5 times an incapacitating one, which weighs twice a non-incapacitating injury).

We used three calculated members: the calculation of the `RiskScore` is determined by weighting different injury severities and combining them into a single metric, then we select the highest RiskScore (`CrashTypeWithMaxRisk`), and then we compute the Damage of the top crash type (`TotalDamageRiskiestCrashType`). We used only injury-related measures (we did not consider NO INDICATION and UNKNOWN because they do not indicate the presence of the injury).

Since the CRASH TYPE attribute contains only two values, only one of which is strictly injury-related, as is visible below only "INJURY AND/OR TOW TWO TO CRASH" turns out to be the top crash type. Therefore, to further study and verify that the query is working properly, we decided to perform the same kind of analysis using FIRST CRASH TYPE. It can be seen that ANGLE turns out to be the riskiest first crash type for as many as four years, based on the reported data.

```
WITH
    MEMBER RiskScore AS
        (5 * [Measures].[INJURIES FATAL]) +
        (1 * [Measures].[INJURIES INCAPACITATING]) +
        (0.5 * [Measures].[INJURIES NON INCAPACITATING]) +
        (0.1 * [Measures].[INJURIES REPORTED NOT EVIDENT]),
        FORMAT_STRING = "#.##"
    MEMBER CrashTypeWithMaxRisk AS
        TOPCOUNT(
                [Crash Report Dimension].[CRASH TYPE].[CRASH
                    ↪ TYPE],
                1, RiskScore).ITEM(0).NAME
    MEMBER TotalDamageRiskiestCrashType AS
        SUM(FILTER(
                [Crash Report Dimension].[CRASH TYPE].[CRASH
                    ↪ TYPE],
                RiskScore = MAX([Crash Report Dimension].[CRASH
                    ↪  TYPE].[CRASH TYPE], RiskScore)
        ), [Measures].[DAMAGE]), FORMAT_STRING = "Currency"
SELECT { CrashTypeWithMaxRisk, TotalDamageRiskiestCrashType,
    ↪ RiskScore } ON COLUMNS,
    NONEMPTY([Date Dimension].[YEAR].[YEAR]) ON ROWS
FROM [GROUP_8_CUBE]
```

	CrashTypeWithMaxRisk	TotalDamageRiskiestCrashType	RiskScore
2014	INJURY AND / OR TOW DUE TO CRASH	$10,961.95	.5
2015	INJURY AND / OR TOW DUE TO CRASH	$5,775,712.38	73.8
2016	INJURY AND / OR TOW DUE TO CRASH	$33,899,590.76	248.3
2017	INJURY AND / OR TOW DUE TO CRASH	$139,877,137.63	473.3
2018	INJURY AND / OR TOW DUE TO CRASH	$250,806,231.74	730.3
2019	INJURY AND / OR TOW DUE TO CRASH	$6,271,297.52	85.9

Figure 3.1: Results with CRASH TYPE

	CrashTypeWithMaxRisk	TotalDamageRiskiestCrashType	RiskScore
2014	ANGLE	$10,961.95	.5
2015	REAR END	$16,604,775.80	73.8
2016	ANGLE	$31,063,362.11	248.3
2017	ANGLE	$68,809,892.00	473.3
2018	ANGLE	$107,479,062.57	730.3
2019	TURNING	$3,191,855.22	85.9

Figure 3.2: Results with FIRST CRASH TYPE

3.3 PowerBI Dashboards

Assignment 9

The final part of the project focuses on Power BI, specifically on Assignment 9, which examines the geographical composition of our data. In this task, we created a dashboard showing the total damage costs for each vehicle based on their location. We decided that the best approach would be a map visualization using the following parameters: DAMAGE LATITUDE, LONGITUDE and VEHICLE TYPE We also identified additional important geographical factors, for instance, WEATHER CONDITION reveals that most accidents occurred under clear skies, while STREET DIRECTION and ROAD DEFECT indicate that crashes were most frequent when traveling west and that there were generally no defects in the road. To provide further detail about road conditions, we included SURFACE CONDITION,which showed that, in most cases, the status of the road was reported as unknown. An interesting aspect of the geographical analysis is the TRAFFIC CONTROL DEVICES variable, which provides insights into road conditions. As the data shows, most crashes occurred either without any traffic controls or at intersections controlled by a traffic signal.

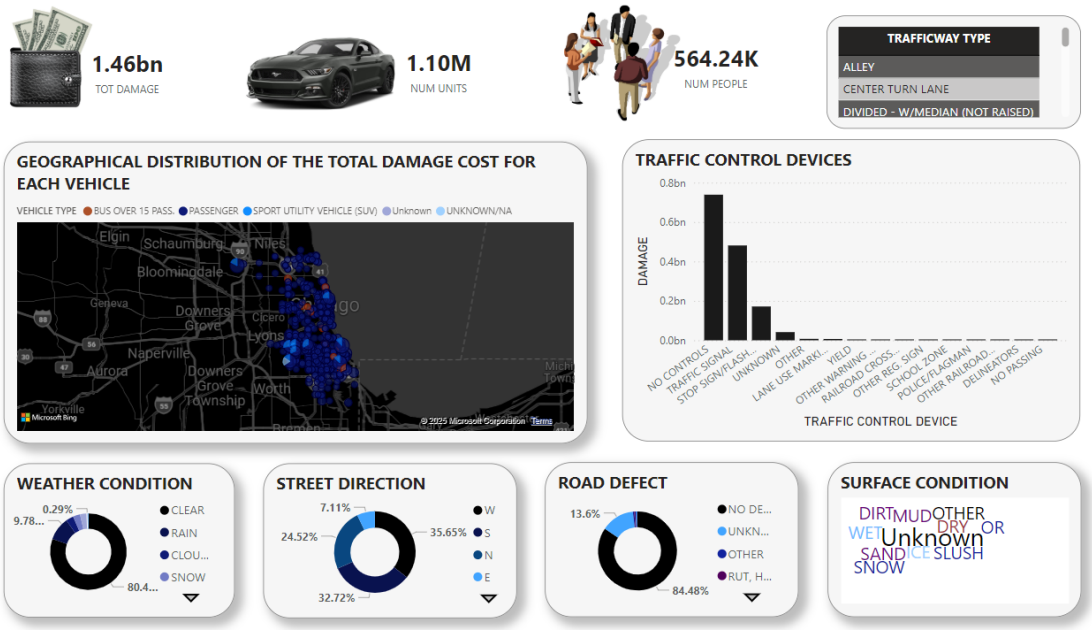


Figure 3.3: Dashboard for Assignment 9 - Focus on the geographical distribution of the total damage costs for each vehicle category. In the figure, the map has been filtered for DAMAGE greater than 100000 to make the visualization cleaner.

Assignment 10

This dashboard analyzes crash data focusing on information related to the **streets**: we inserted as key metrics the *total number of crashes*, *total number of units involved*, and *total damage amount*. In this screenshot all the data are filtered by **year** (2017, 2018 and 2019) and **injury classification** (FATAL and INCAPACITATING) while viewing all weather conditions. A map visualizes crash hotspots geographically, while a line graph tracks daily crash patterns for the top streets. We can extract some insights: Pulaski RD has the highest number of reported crashes with peaks on days 12 and 26. Contrary to what might be expected, most of the amount of damage is related to a DRY condition of the road surface, so based on the available data the fact that the road is WET does not seem to be a major cause.

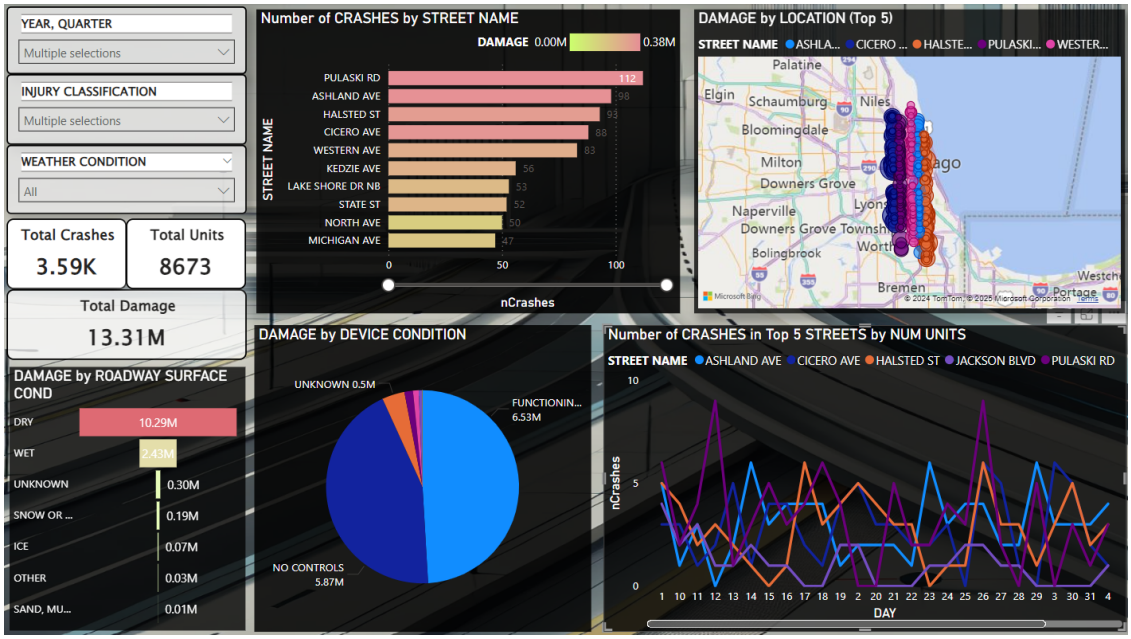


Figure 3.4: Dashboard for Assignment 10 - Focus on street-related information

Assignment 11

The dashboard detailed below presents data focused on information about individuals involved in a crash. Starting from the top-left corner, there is a ring chart that illustrates the percentage of total damage by person type. In this visualization, drivers and passengers have been filtered out because they account for the majority of the damage. This allows us to focus on all other person types. In the bottom-left corner, there is a tabular representation of the data, designed to provide a quick overview of information such as person type, physical condition, injury classification, and the associated damage category. In the top-right corner, a treemap is included to analyze damage by injury classification. The final visualization is a bar chart that displays damage by age group. Lastly, all the information can be filtered based on the primary contributory cause of the crash and the weather conditions.

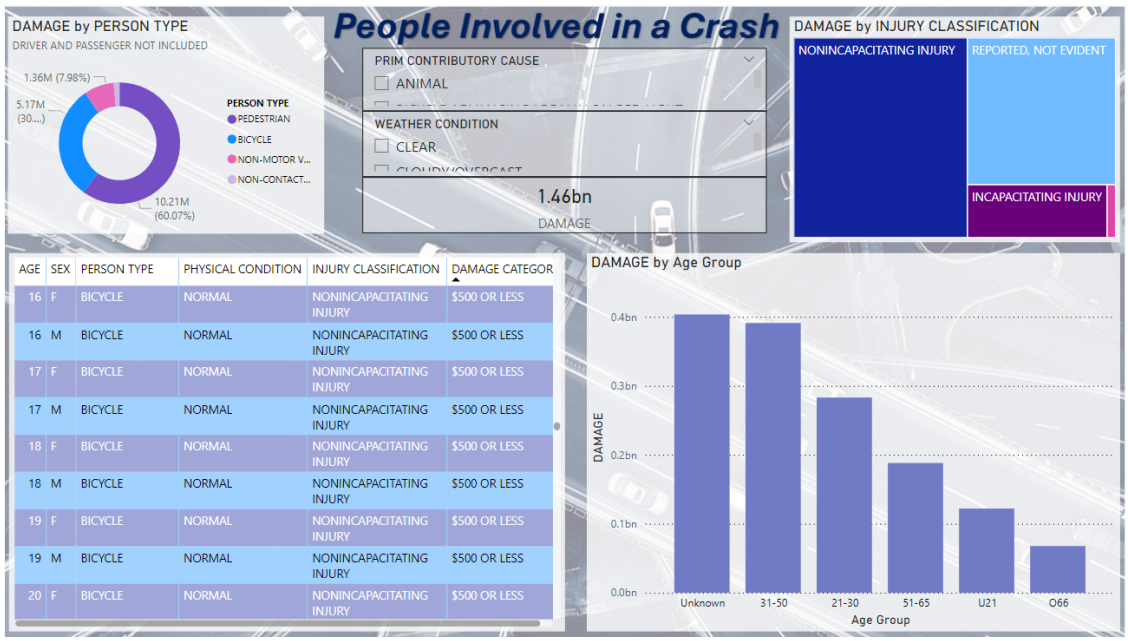


Figure 3.5: Dashboard for Assignment 11 - Focus on people involved in a crash information

Bibliography

[1] *Boundaries - Police Beats (current) — City of Chicago — Data Portal*. https://data.cityofchicago.org/d/aerh-rz74.

[2] *Traffic Crashes - Chicago*. kaggle.com/datasets/isadoraamorim/trafficcrasheschicago.