UNIVERSITÀ DI PISA

DEPARTMENT OF COMPUTER SCIENCE

MASTER'S DEGREE IN DATA SCIENCE & BUSINESS INFORMATICS

LABORATORY OF DATA SCIENCE

Analysis of a Business Intelligence process

Academic year 2020/2021

Authors:
Tommaso CAVALIERI
Andrea FEDELE

December 31, 2020

Introduction

The term Business intelligence (BI) comprises the strategies and technologies used by an enterprise for the data analysis of business information: BI technologies, in fact, provide historical, current, and predictive views of business operations. In this report will be presented the solutions to a set of assignments corresponding to a Business Intelligence process such as data integration, construction and querying of an online analytical processing cube, dashboard development and reporting. BI technologies can handle large amounts of data to help identify, develop, and create new strategic business opportunities, with the aim of allowing an easy interpretation of such big data. In the analysis carried out for this project, such data was not already properly structured, thus a data warehouse was created first and such process will be presented in Chapter 1, while the operations performed on it will be described in Chapters 2 and 3.

Chapter 1

Creation of the database and performance of basic operations

Business Intelligence applications usually exploit data gathered from a data warehouse, which contains a copy of analytical data that facilitate decision support. In this chapter will be described step by step the building and population of such data structure, starting from some datasets originally available in **csv** format.

1.1 Definition of the database schema

The first step of such process was thus to build a data warehouse in order to later exploit it to perform some business analysis. To complete such task, the software **SQL Server Management Studio** was used and the database schema shown in Figure 1.1 was created. This data source was given the name **Group12HWMart** and will henceforth be referred to as such. The information regarding the attributes of the tables in such schema was retrieved by simply opening the aforementioned original files in **csv** format and looking at their characteristics, i.e. their type (strings, integers etc.). The number of characters reserved for the string attributes was set equal to the length of longest value taken by such attribute increased by 15. For almost every table we were able to select one attribute as primary key, represented by the little key symbol next to its name in the schema. However, for the tables ***GPU_ Sales, CPU_ Sales*** and ***RAM_ sales***, it would have been necessary to concatenate *GPU_ code, CPU_ code* and *RAM_ code* to obtain a superkey that would uniquely identify every record. It was therefore preferred to insert a new identification number in each of these tables, respectively *GPU_fact_ id, CPU_fact_ id* and *RAM_fact_ id*, which all function as primary key. Please note that even though the 'code variables' could not be used as primary key, they still were not eliminated since they were a foreign key useful for connecting fact tables to the product tables, i.e. *GPU_ code* in the table ***GPU_ sales*** is a foreign key to the table ***GPU_ product***.

1.2 Generating the fact tables

The second task was to retrieve the sales data to populate the database just designed and, in order to do so, a Python program was written that would split the file ***fact.csv*** into three separate tables. With the help of the csv library, such file was opened and then, after

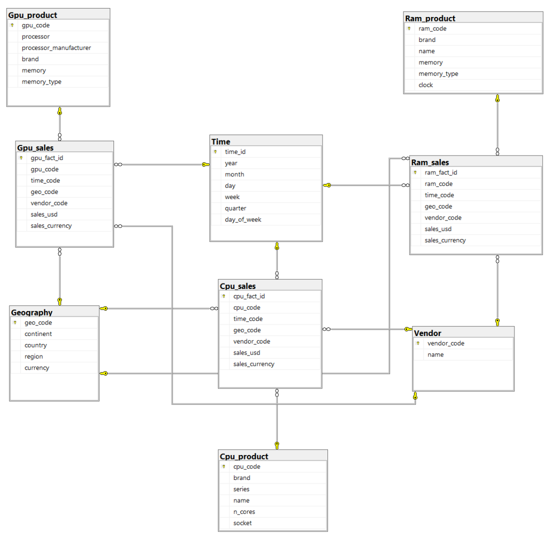


Figure 1.1: Database schema

opening 3 distinct files (one for each table to be generated), the deployment of the data was based on the values of the dummy variables *GPU_code, CPU_code* and *RAM_code*. In fact, such attributes were binary, simply indicating whether the record referred to a GPU, CPU or RAM product; when one of them took value 1, the others must be equal to 0. The ID's format was converted from float to integer and a 'flag' was used for each table to insert the appropriate header line in the 3 new files. Such flag was initially set to 0 and stayed unchanged until the first record was written in the file it referred to, then it was switched to 1 to avoid to re-insert the same header. Please note that this task was solved without implementing the Pandas library, as explicitly required.

1.3 Populating the database schema

After completing the split into GPU, CPU and RAM tables, new information was extracted from the date of purchase of each record: the day of the week (Monday, Tuesday etc.) was obtained thanks to the datetime library while the quarter of the year was nothing more than a derived column which took value Q1 from January to March, Q2 from April to June, Q3 from July to September and Q4 from October to December. Once these attributes were calculated, the population of the database began. Another Python program was written in which, thanks to a dictionary structure, a mapping was created between the database tables and the files that were meant to fill them; moreover, two customized functions were defined: one returning a string like format that could be exploited to concatenate both features and values of each row and another that would later concatenate them in a query like format compliant with the SQL syntax. Afterwards, by using the proper credentials (username and password) we were given the authorization to establish a connection to the DBMS. At this point, for each file in the mapping dictionary, a file and a cursor were opened and then for each line in the original csv files, an INSERT INTO query was generated and then committed to populate the table, before closing the cursor and the connection. Please notice that the files GPU_sales, CPU_sales and RAM_sales were not populated in the same run, but one at the time, following the procedure just described; such trick was necessary due to the big dimensions of such files.

Chapter 2

Business questions resolution

In this chapter will be described the solutions of other tasks, which consisted in answering some business questions regarding the information available in the database previously created. Such assignments have been faced using **Sequel Server Integration Services (SSIS)** and leaving the computation on the client side. In fact, the goal was to implement the solution to such question by only creating apposite flows of nodes in **Visual Studio**, without exploiting any SQL command. Each of the following paragraphs shows the business question in the title and the description of its solution in the body.

2.1 Assignment 0 - *(List) For every vendor, the brand of CPU ordered by sales.*

In order to return such list, we decided to consider the *CPU_sales*, *Vendor* and *CPU_product* tables. The first step was to fetch data from the *CPU_sales* table by means of an OLEDB Source node. Then the vendor's name and the CPU's brand were retrieved by means of two LookUp nodes, respectively from the *Vendor* and the *CPU_product* table. Once all the data were available, an Aggregation node was used in order to apply the GROUP BY operation on both *vendor_name* and *CPU_brand* attributes, summing up the *sales_usd* value. This process allowed us to have the total amount of sales grouped by both vendor and CPU brand. In order to sort this data by sales, we applied a Sorting node that would sort in a descending order on the *sales_usd* field, as requested by the assignment, and on an ascending way on the *vendor_name*, for layout purposes. The resulting list has then been written on a text file called **Assignment0.txt** by means of a Flat File Destination node, including also the header. Please notice that the attribute *sales_usd* was preferred to *sales_currency* because the former presented all the sales values in the same currency, US dollars ($), while the latter showed their value in the original currency, which changes from one record to another, thus preventing us to apply aggregation functions such as the sum to it. Such reasoning was kept through-out the whole process, in fact when sales will be investigated afterwards, we will always be referring to *sales_usd*. In Figure 2.1 it is possible to appreciate the flow just described.

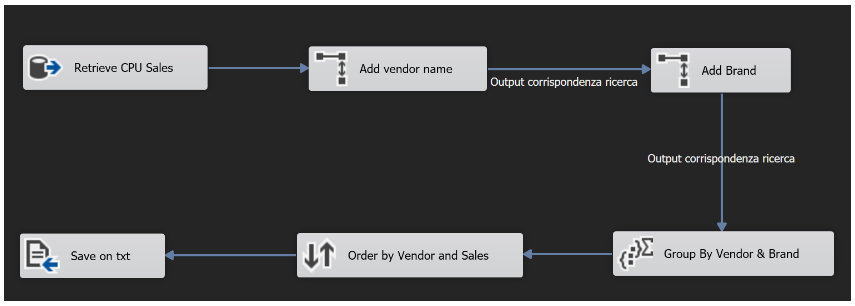

Retrieve CPU Sales
Add vendor name
Output corrispondenza ricerca
Add Brand
Output corrispondenza ricerca
Group By Vendor & Brand
Order by Vendor and Sales
Save on txt

Figure 2.1: Flow of the Assignment 1 solution

2.2 Assignment 1 - *List all the vendors that are franchised to a brand of RAM. A vendor is said to be franchised if more than 60% of his sales come from the same brand of product.*

For the purpose of answering such question we focused on the *RAM_sales*, *RAM_product* and *Vendor* tables. The main source of data was the *RAM_sales* table from which all the rows were fetched by means of an OLEDB SOURCE node. Right after the main fetch, the RAM brand and the vendor name were retrieved by means of LOOKUP nodes, respectively from the *RAM_product* and the *Vendor* tables. Once all these columns were collected, we decided to split the flow into two branches, exploiting a MULTICAST node. While on one branch we applied a GROUP BY only on the vendor's name, summing up the values of *sales_usd*, on the other one we did the same but grouping also by the RAM's brand. These two branches were then merged together by means of a MERGE JOIN node, after having properly sorted both of them by the same attribute. Such operation allowed us to keep every useful column: the vendor name, the RAM brand, the total sales for each vendor and the sales for every brand. Since we were interested in checking the ratio between total sales and the ones of a specific brand, we derived a column by means of the DERIVATIVE COLUMN node, namely *sales_percentage*, in which such ratio was stored. Once computed such value, we filtered out only the rows satisfying the "being franchised" property, exploiting a CONDITIONAL SPLIT node, hence keeping only the records with a *sales_percentage* equal or higher than 0.6. Those records were later written in the **Assignment1.txt** file by means of a FLAT FILE DESTINATION node, where, headers included, are reported the vendor's name and the RAM's brand to which it is franchised. The flow of nodes required to complete such task is shown in Figure 2.2. Please notice that, as required, the conditional split gives as output only the vendors that are actually franchised to some brand and therefore end up being reported in the text file.

2.3 Assignment 2 - *List the memory type of GPUs that sold more than the average sales for GPUs, for each month.*

Aiming to fulfil this last chore, we decided to work on the *GPU_sales*, *GPU_product* and *Time* tables. The main selected data source was the *GPU_sales* table, from which all

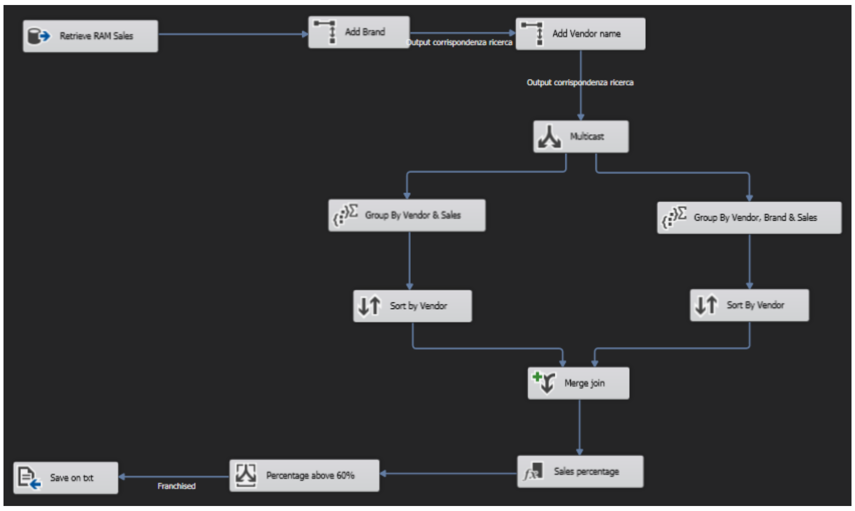


Figure 2.2: Flow of the Assignment 1 solution

the rows were fetched by means of a OLEDB Source node. Right after such data were fetched, the memory type, month and year were also retrieved thanks to two distinct LookUp nodes, respectively from the *GPU_product* and *Time* table. Once all these columns were recovered, we split the flow into two branches with a Multicast node: while on one side we applied a grouping only by year and month, averaging the *sales_usd* value, on the other one we included the GPU memory type as an additional attribute of the GROUP BY. After sorting both branches on the same attribute, they were merged together by means of a Merge Join node. Such operation allowed us to keep all the useful columns: month, year, memory type and both values of the average sales, with and without the additional grouping by memory type, which were renamed respectively (*avg_sales*) and (*sales_usd*). To conclude, we filtered out the rows satisfying business requirements by means of a Conditional Split node, considering only the records where the value of *sales_usd* was higher than the one of *avg_sales*. The filtered records were then written on the **Assignment2.txt** thorough a Flat File Destination node, including also the header. The printed values were the year and the month in which the reported memory type sold more than the average sales. Please notice that, as required, thanks to the conditional split only the memory types that have sold above the average are reported in the text file. Figure 2.3 shows the flow defined to solve this business question.

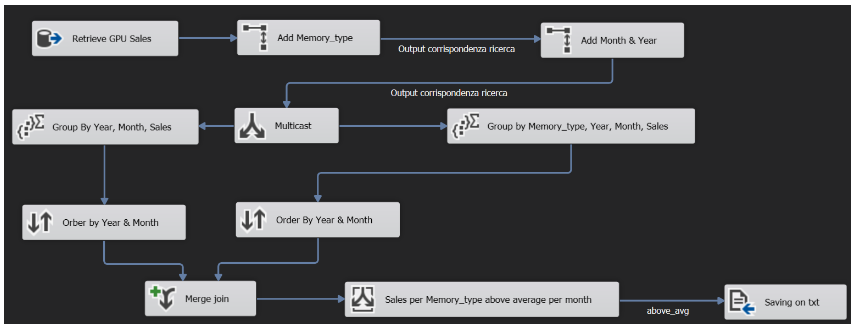

Retrieve GPU Sales
Add Memory_type
Output corrispondenza ricerca
Add Month & Year
Output corrispondenza ricerca
Group By Year, Month, Sales
Multicast
Group by Memory_type, Year, Month, Sales
Orber by Year & Month
Order By Year & Month
Merge join
Sales per Memory_type above average per month
above_avg
Saving on txt

Figure 2.3: Flow of the Assignment 2 solution

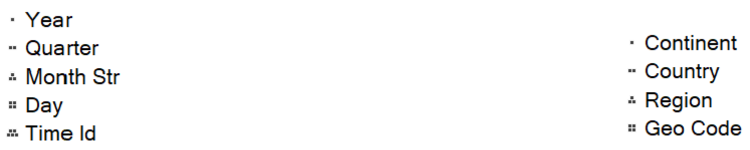
Chapter 3

Creation and exploitment of a datacube

In this last chapter we will present the final tasks of the analysis, which include the generation of a datacube and its interrogation through appropriate queries. Finally, some visual representation will also be presented in the form of dashboards.

3.1 The cube

After establishing a connection to the database Group12HWMart with **Visual Studio**, a view of such schema was created, containing the following tables: *CPU_sales, CPU_product, Time, Geography* and *Vendor*. Starting from such view, an OLAP cube was later generated by selecting the CPU sales as measure and all the other existing tables as dimensions. Two distinct hierarchies were then built, in order to manage both the temporal and spatial dimensions of the data, namely *TimeByDay* and *GeographyByRegion*, whose compositions are shown in Figure 3.1.



(a) TimeByDay (b) GeographyByRegion

Figure 3.1: Hierarchies

3.2 MDX queries

Once completed the designing of the cube, it was possible to solve some business questions, but this time exploiting **Sequel Server Analysis Services (SSAS)**, in particular using MultiDimensional eXpressions (MDX) in **SQL Server management studio**.

3.2.1 Assignment 1 - *Show the ratio between weekdays sales and weekend sales for each month and CPU brand.*

In order to answer this query, both the weekday's and weekend's sales were computed as two distinct MDX CALCULATED MEMBERS by means of the MDX SUM function. These two members were computed exploiting the Day of Week attribute, considering as weekday sales the ones purchased between Monday and Friday and as weekend sales the ones from Saturday and Sunday. Once these two members were derived, we computed the ratio between them and stored it using another MDX CALCULATED MEMBER. The ratio was then shown in a column, while the month and the CPU brand were listed in the rows.

3.2.2 Assignment 2 - *For each vendor, show the difference between the total sales of each month and the total sales of the previous month.*

For the purpose of answering such query, the difference between the total sales of each month and the previous one was saved in a MDX CALCULATED MEMBER. Such value was obtained by computing the difference between the *sales_ usd* measure and the same measure from the previous month, which could be retrieved with the proper use of the *TimeByDay* Hierarchy and the MDX LAG function. Such function was applied on the CURRENTMEMBER, which was the single month, and by setting 1 as the parameter, it accessed the previous one. The difference was then shown on the columns, while the vendor name along with the month and the year were listed in the rows.

3.2.3 Assignment 3 - *For each country show the CPU series with the highest total sales and the sales ratio between that CPU series and the total sales for that country.*

Aiming to solve this last query, the first computation was made to obtain *sales_per_ country* as an MDX CALCULATED MEMBER, by grouping the *sales_ usd* measure by country. Once the sales per country were computed, the percentage of sales of every CPU series in a given country was stored in another MDX CALCULATED MEMBER as the ratio between *sales_ usd* and *sales_per_ country*, with the format string property set as "Percentage". Once this ratio was available, we used the MDX GENERATE function to apply a specific operation (MDX TOPCOUNT) to every member of given set, joining the resulting sets by union. By doing so, the series with the highest sales was individuated for each country and then reported with the following schema: the pair country-series on the rows, *Sales_ Usd*, *Sales_per_ country* and the percentage on the columns.

3.3 Dashboards

The last step of the process described in this report was to create some dashboards in order to give the user a deeper insight on some dimensions of the data, namely the temporal and geographical ones. The software exploited for creating these panels were **Power BI**, for the first two assignments, and **Microstrategy** for the last one.

3.3.1 Assignment 4 - Temporal view

The first goal was to create a dashboard that would show how sales changed over time and in order to do so a grouped bars chart was created, highlighting the distinction between weekday and weekend sales, giving also the user the opportunity to see the sales behavior for different time granularity with a drill-down operation, exploiting the *TimeByDay* hierarchy. An area chart showing the behaviour of total sales through time with respect to the CPU brand was also included, along with two distinct line chart showing the same measure, but this time with respect to the various vendors. This last graph was split in two because putting all the vendors together would have made it less clear, because of the differences in terms of orders of magnitude of the sales for each vendor. Please notice that by selecting a specific period in the bar chart, such filter will also be applied to the other graphs, thus resulting in a dynamical environment. The described dashboard is shown in Figure 3.2.

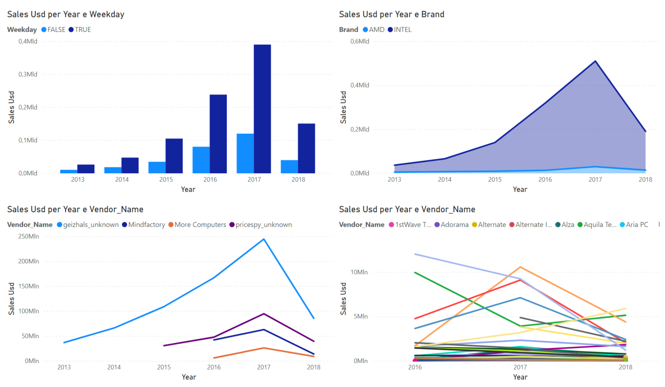


Figure 3.2: Temporal Power BI Dashboard

3.3.2 Assignment 5 - Geographical view

The second dashboard created with Power BI (Figure 3.3), showed the total sales, this time both meant as total amount of sales and as number of distinct purchases, and their geographical distribution. It includes two maps, a coloured one for the number of purchases and a classic one for the total sales, each of them linked to a specific doughnut chart, which shows the distribution of the two measures at different geographical granularity. Please notice that the colour variation for the different areas on the map on the left is defined to depend on the value of the explored attribute, that is the count of distinct sales.

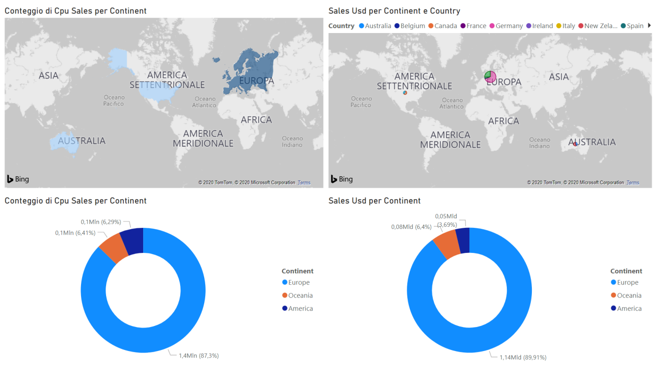


Figure 3.3: Spatial Power BI Dashboard

3.3.3 Assignment 6 - Open task

This last dashboard was implemented, differently from the others, with **MicroStrategy Workstation** and by means of the Dossier Analysis Tool. The goal for this implementation was to give the user a complete view both of the temporal and geographical dimensions of the data, by exploiting some attributes that were not considered in the previous dashboards. Unfortunately, Microstrategy software did not allow to access directly the defined cube and exploit its hierarchies, but it was still possible to explore the data in it by uploading the desired tables. The created panel, plotted in Figure 3.4, shows the *Sales By Country* both on a map, where the country color gradient depend on the sales attribute value, and in a ring chart. The *Sales By CPU Series* is an area chart plotting the total sales with respect to the CPU series attribute, while the *CPU Product and Sales by N.cores* includes two line charts, showing the total sales and the number of CPU products that were sold for any given number of cores. Finally, the *Sales By Weekday* view is a grouped bar chart showing the total sales per each year, divided and coloured according to the day of the week in which whey were purchased. Please notice that in such visualization the first day of the week was *Sunday*. Two main filters were also inserted at the top of the page to give the user the opportunity to refine all the graphs simultaneously, according to his/her specific points of interest. In fact, *Filter By Year* allows the user to select a specific time range, while *Filter By Country* selects the countries whose data should be plotted in the dossier's graphs. By doing so, we were able to synthesize the most intriguing facets in one single page, giving the user the opportunity to investigate, with just a couple of clicks, the sales behaviour with regard to the two most important dimensions in our data: time and space.

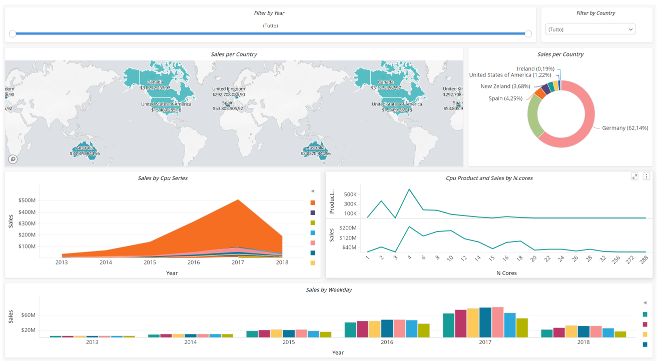


Figure 3.4: Microstrategy Dashboard