UNIVERSITÀ DI PISA

Laboratory of Data Science: A Written Report

Building a Data Warehouse, Answering Business Questions Using SSIS & Building and Querying an OLAP Cube

Guiducci Federica (600310)
f.guiducci2@studenti.unipi.it

Ilic Ema (602796)
e.ilic@studenti.unipi.it

Olivotto Valentina (600009)
v.olivotto@studenti.unipi.it

Laboratory of Data Science (664AA), Anno accademico 2020/2021

Indice

1 Assignment I: Building a Data Warehouse

The team was provided a zip folder containing seven csv files and asked to build a Data Warehouse in SQL Server Management Studio on the university's remote server 'apa.di.unipi.it' taking into account the guidelines provided in the three tasks. Starting from the files provided *(cpu, fact, geography, gpu, ram, time, vendor)*, a new set of tables was isolated, as suggested by the tasks (**Gpu_product, Ram_product, Cpu_product, Gpu_sales, Ram_sales, Cpu_sales, Geography, Time, Vendor**). The lack of missing values in each output table was confirmed with the help of the *Missingvalues.py*, which pointed out correctly all the missing values in the initial **fact** table. Indeed, only one of the attributes *gpu_code*, *cpu_code* and *ram_code* takes on a non-null value in a given row.

1.1 Task 0: Create a logical database schema in SSMS

The tables and their respective attributes and types are represented in Figure 1. Even though initially all the data types were strings, after careful visual analysis of the nature of the values, some were casted to integers (e.g. *n_cores* of **Cpu_product**) and floats (e.g. attribute *sales_usd* in **Ram_sales**). All the key columns are of type integer and have an active 'Identity' setting, except for the *time_code* (primary key in the table **Time**) which was set as a string rather than a datetime type precisely because of its identifying nature. Moreover, the information on the actual date (year, month, day) are represented in the remaining columns of the **Time** table. Final remark worth noting is that the attribute *gpu_code* was represented as a float-string (e.g. '4.0') in the initial **gpu** table, and was thus casted to an integer in the output tables.

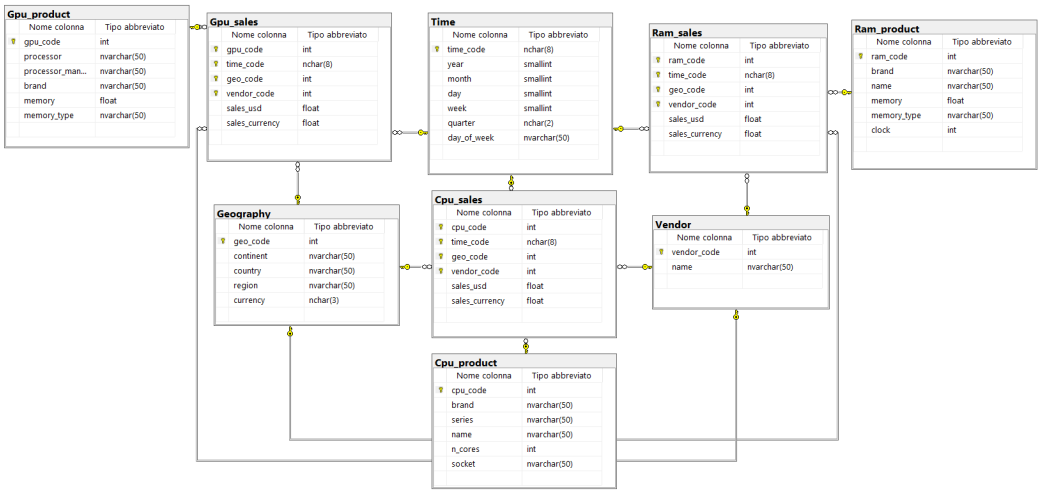

Figura 1: Data Warehouse Schema

1.2 Task 1: Split the Fact table in three, without using Pandas library

The Task 1 was to split the **fact** table into three distinct tables (**Cpu_sales, Gpu_sales** and **Ram_sales**), each of them corresponding to one of the product lines without using the Pandas library. The task was achieved with the *Fact_Table_Division.py* program where the csv library was used for reading and writing the files. At the end of the program the sum of the lines of all the three output tables was compared to the length of the initial **fact** table to verify the correctness of the split.

1.3 Task 2: Populate the database and modify the table Time

The first part of this task provided for modifying the **Time** table in a way so that the *day_of_week* and *quarter* should be derived from the date, which was tackled in the *Time_Changing.py*. It is worthwhile mentioning that the *calendar* library was of great help in converting the dates to the days

of the week, whereas the *quarter* attribute was retrieved from the *month* attribute. After that the final task was to fill out the database Group02WHMart with all the tables shown in Figure 1. This was achieved with the *Upload.py* file, which contains a generalized *upload* function for uploading any table to the Group02WHMart database (as an example, we left the upload of **Time** table at the end of the program). Additionally, the codes for uploading the largest tables (**_sales**) were run simultaneously to speed up the process. Here is where the connection with the server is established and a generic query string for insertion ('INSERT INTO ...') creates a table with appropriate column names and as many question marks as there are columns in a given table. Later on an iterator fills out the tables in the database row by row with the input tables. Another important function of this file worth noting is the *casting* function, its scope being the smart conversion of any attribute into a corresponding type (float, integer or string), as in the initial files, all the types were strings.

2 Assignment II: Answering Business Questions in SSIS

The second assignment was to answer some business questions using the data warehouse created in Assignment I using only the SSIS tool in Visual Studio, with computation on the client side (i.e. without writing queries). To skip the redundancies in the report, it is important to note that for all the tasks, the initial and final steps are the the OLE DB Source node, the Lookup node(s) and the File Flat Destination node. Before getting into the solutions, it is worthwhile noting that every time a lookup and the conditional split node were used, the results verifying the condition were kept. Additionally, only the relevant attributes were kept in each step in order to reduce the computational cost. The results were exported with the generic name *Task_n_output.txt*, where n refers to the ordinal number of the task. The solutions can be found in the *Group2_SSIS.zip folder*.

2.1 Task 0: For every year, the brand of gpu ordered by sales

Upon accessing the **Gpu_sales** table, the Lookup node was used to join **Time** and **Gpu_product** tables in order to retrieve the *year* and the *brand* attributes. Grouping by year and brand was achieved in the Aggregation node as well as summing the total sales. For aesthetics it was decided to order the results by year ascending, and by *total_sales* descending.

2.2 Task 1: List all the AMD brand cpus that do not have full regional spread in Germany.

For any given country, a product is said to have full regional spread if it was sold in all the regions of that country. Therefore, in this task, it is important to mention the use of the conditional split node to isolate only the "AMD" brand and "Germany" as a country. Another interesting feature of the task is a two-fold Multicast node meant to count the number of distinct regions in Germany and then add the resulting number (16) at the end of every row of the multicast input as a *Region_Count_Germany* coulumn using the Derived Column node's 'Fake Key' trick. Once the join was done on the sorted *fake key*, the result was aggregated in the following way. First a GroupBy operation was done on the *cpu_code*, then Count Distinct was done on the Region (*Region_Count*) and finally, a trick was done using the Minimum function on the *Region_Count_Germany* in order to avoid a multicast loop. This way the count of regions by *cpu_code* was achieved. Finally, in order to list only cpu units which do not have full regional spread, the Conditional Split node was used with the following constraint: $Region\_Count < Region\_Count\_Germany$.

2.3 Task 2: Processor manufacturer yielding the most sales, for each country and year.

A particularity of the solution to the last task of the second assignment was the Aggregation node to calculate the total sales: namely, grouping by country, year and the processor manufacturer, and summing the *sales_usd* as *total_sales*. Another feature of the solution worth noting is the two-fold Multicast operator. On the left side there is an Aggregation node for calculating the maximum of the

total_ sales called *max_ total_ sales* while grouping by year and country. Both the left and the right side are sorted on the year, country and total sales and then joined on the same attributes (except for *total_ sales* which was joined with *max_ total_ sales*), while keeping also the *processor_ manufacturer* attribute.

3 Assignment III: Building and Querying an OLAP Cube & Reporting

3.1 Task 0: Build a multidimensional OLAP Cube

The OLAP cube was built in MS Visual Studio using the SSAS integration from the following tables from a Data Warehouse created in the Assignment I: **Gpu_product, Gpu_Sales, Vendor, Time** and **Geography**, as provided for in the task. There are several aspects of the process worth mentioning. First of all, *Month* attribute in the Time table was initially comprised of numbers 1-12. This attribute was used as a sorting key in the 'Advanced Options' menu for the new attribute created, *Month Name*, which takes on values from 'January' to 'December'. On the other hand, in order to sort the *Day_ of_ Week* attribute, another attribute (*Order_ Day*) was created in order to serve as a sorting key. Thus, the *Month* and the *Order_ Day* attributes were hidden using the 'AttributeHierarchyVisibility' option from the same menu. Thus, three hierarchies were created, two concerning the **Time** table ('Time_hierarchy' including *Year, Quarter, Month Name, Time Code* and 'Week Hierarchy', including *Year, Week, Time Code*), and one in the **Geography** table (Geography Hierarchy: *Continent, Country, Region*). The *Week* attribute was not included in the 'Time Hierarchy' due to the incongruities which occasionally arise when a week stretches across two months. For that reason, the Week Hierarchy was created. Note that the ID attribute (*GeoCode*) isn't included in the Geography hierarchy because a functional dependency was imposed in this hierarchy. It is also worth noting that the *Sales USD* and *Sales Currency* were formatted as currencies.

3.2 Tasks 1-3: Querying the cube with MDX

Task 1 asked for the total sales for each country and vendor as well as the grand total of the continent. The results are portrayed in Query1 which can be found in the 'MDX_Group2.mdx' file. There were several ways to visualize the grand total with respect to the continent, and thus it was decided to visualize it as a new column which takes on 3 different possible values, depending on the continent the given row is concerning. Task 2 asked for the total sales USD, total Sales Currency, total difference between the two for each month and the running difference for each year for Germany. Both of the new columns were added using the 'member' function; however, the running difference was achieved with the aid of 'periodstodate' function. Finally, the Task 3 called for showing the top 5 GPU brands w.r.t to the monthly average sales for each region in Europe. This was achieved by calculating the overall number of months in the period observed between March 2013 and April 2018 using 'member' function, and dividing the total sales of each vendor in each region by the total number of months constant in order to obtain the average monthly sales. The choice was justified by the fact that the product in question is not a seasonal one. The function 'Generate' was used to loop over regions whereas the 'Top Count' function selected only the top five brands per each region according to their monthly average sales.

3.3 Tasks 4-6: Reporting through Dashboards

Each of the final three tasks was solved using both the 'Microstrategy' and 'Power BI' tool. For the sake of conciseness of the report, only the non-obvious features of the reports are going to be mentioned here. For the reference, the Task 1 called for creating a dashboard that shows how sales change for different granularity levels of time; Task 2 asked for showing the geographical distribution of sales and of the number of products purchased; and Task 3 provided for creating an interesting dashboard of team's choosing. Thus, it was decided that in Power BI the Task 3 is going to comprise European

sales with a focus on brand and processor manufacturer, whereas in Microstrategy the focus is going to be the sales of PNY brand because it achieved the highest sales.

Microsoft Power BI

This tool had a direct access to the Datacube created in Task 1 of the Assignment 3. Thus, it was possible to take advantage of the hierarchies created in the cube through the 'Drill Down' function of the plots. In the document Group2_Dashboard.pbix three pages can be observed, each corresponding to the specific task at hand. On the page 'Task 2', two maps can be observed, one being a filled map which represents the count of GPU sales by country, and another map with bubble markers representing the sales expressed in USD. A particularity of the page 'Task 3' is that the color rules of a Map plot in the 'Format' > 'Data Colors' menu were manually set into three bands: When the value of sales is lesser or equal than 10,000,000, between 10,000,000 (included) and 100,000,000, and between 100,000,000 (included) and 300,000,000, with each of value bands being assigned a deeper shade of green as the sales increase.

Microstrategy

The demo version of the Microstrategy tool did not have an option of direct access to the data cube. Therefore, the relevant tables used in the cube were imported from the data warehouse created in the Assignment 1. The data was prepared before importing so that the attributes in the table. *Continent, Country* and *Region* were converted to Geographical attributes with right click on the *Attribute name*>'Define Geography'. However, it is worthwhile noting that Microstrategy did not recognize *Continent* and *Region* attributes as Geographical attributes later on when applied on a map, as there was no option of setting an attribute as a 'Continent' or a 'Region'. Additionally, in the 'Prepare Data' interface, the software automatically set ID attributes (*GeoCode, Time Code*, etc) as metrics instead of attributes, which was fixed manually. Finally, data was imported as an 'In-Memory Dataset'. An interesting feature from a Data Science point of view was discovered - namely, in the 'Format' menu of an Area Chart there is a drop-down menu allowing for the manipulation of 'Trend Lines', which is where the 'Enable Forecasting' option was found. However, it was not giving reliable results based neither on the five input years, nor on the input years with months. Thus it was decided to create a new attribute using the 'Wrangling' feature called *YearMonth*, its values being created by concatenating the value of *Year* and *Month* attributes. The results were more satisfying and realistic this way. A peculiarity of the reports achieved in Microstrategy, is that a 'Target Visualization' option was set for the maps in Task 5, targeting the remaining charts on the page. This is a default option in Power BI but requires manual setup in Microstrategy.