Laboratory of Data Science Project

Authors:	Craciun Vali, Fabbri Lucia
Course:	Decision Support Systems (Module II)
Master's Degree:	Data Science & Business Informatics

Indice

Introduction

The report's purpose is to provide a concise explanation of how we solved each of the 6 assigned tasks. Our main data sources were the files computer sales.csv and geography.csv. The first contains the primary dataset, presenting a comprehensive table of computer sales spanning from March 2013 to April 2018. Each entry not only details the sales transactions but also provides specifications regarding each PC's CPU, GPU, and RAM configurations. The second one instead enriches the dataset with supplementary information concerning the geographical locations corresponding to each sale. The geography information can be linked to the main fact table with their primary keys.

In figure 1 we can see the datawharehouse schema of reference, which will be used as a guide to solve the very first part of the project, as explained in the next section.

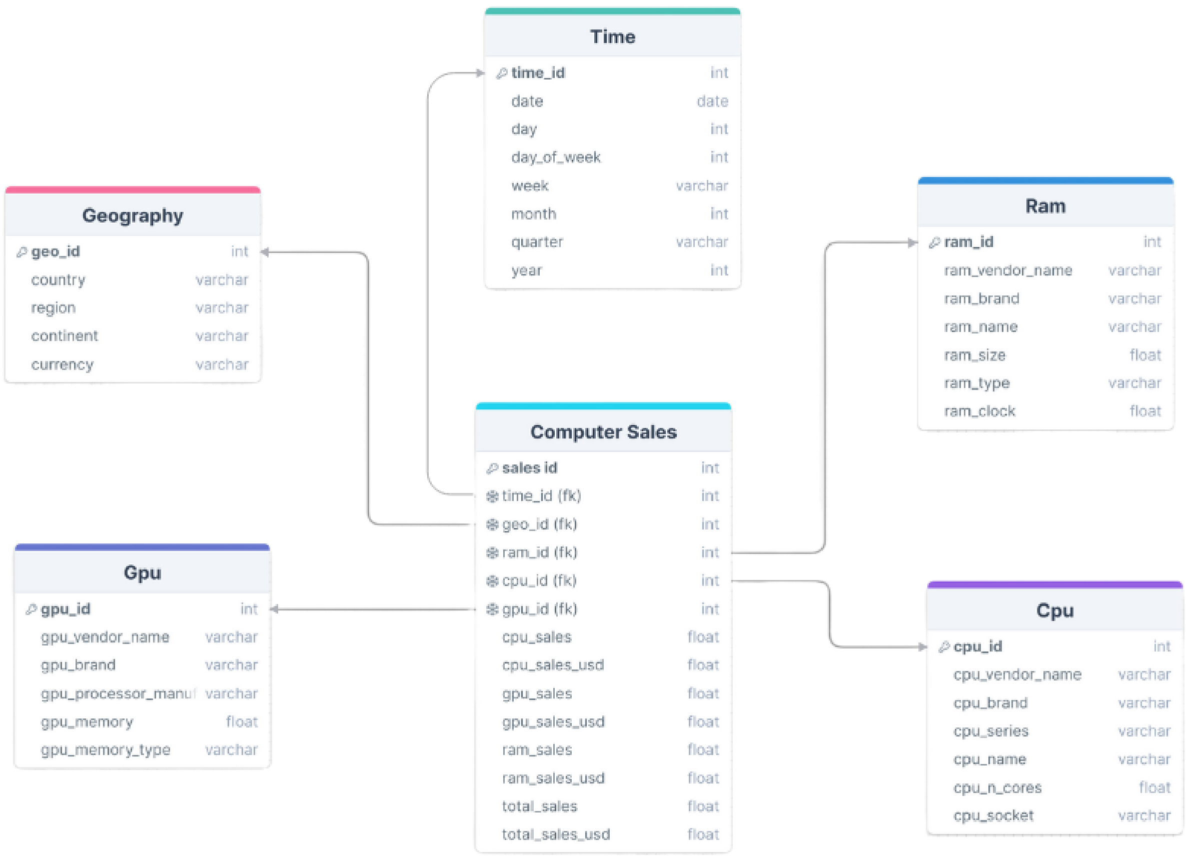


Figura 1: Datawarehouse schema of reference

Part I - Datawarehouse creation

0.1 Assignment 1

As outlined in the guidelines, our initial objective was to create all the necessary tables, starting from the provided datasets, namely computer_sales.csv and geography.csv. To achieve this, we developed a Python script containing several utility functions. We will not dive into the specifics of how each function operates, as the primary approach follows the typical pattern:

```
with (
    open(inputfile, 'r') as input,
    open(outputfile, 'w') as output
):
    reader = ...;
    writer= ...;
    for row in reader:
        modifiedRow = doSomething(row)
        writer.writerow(modifiedRow)
```

What we'll do instead is to explain the main rationale behind their functioning, highlighting how special fields have been obtained.

- **`make_geo(cs`[1]`.csv, geography.csv, geo_attributes, output.csv)`**: Creates the geography table. The main task of this function is to add the *currency* attribute to the already existing geography.csv file. This is achieved by looping over the rows of computer_sales.csv and build a *geo_id-currency* mapping, used later as lookup table for retrieving the right currency, given a geo_id.
- **`make_time(cs.csv, time_attributes, output.csv)`**: Creates the time table. Here we retrieve all the unique *time_code*s from computer_sales.csv and process them using `get_time(time_code)`, as explained below.
  - `get_time(time_code)`: Retrieves all necessary information from each *time_code*. Since each *time_code* was formatted as "yyyymmdd", we could slice this string to extract the year, month, and day. From this information, we derived the attributes *day*, *day_of_week*, *week*, *month*, *quarter* and *year*.
- **`make_fact(cs.csv, [dim.csv], fact_attributes, output.csv)`**: Creates the fact table, i.e., `computer_sales`. This function leverages the previous crafted dimensions treating

[1]abbreviation of computer_sales.csv

them as lookup tables to retrieve the corresponding ids (foreign keys). So, for each row in computer_sales.csv, we split the row according to the various attributes of each dimension and look for the corresponding id of each split row for each dimension. We then write on the output file only the id as foreign key for each dimension. For what concerns the measures, all the *..._sales* fields were already present in the original file, while the *..._sales_usd* fields were missing. To retrieve these values, we utilized the `currencyconverter` library[2]. This process is executed in the `make_fact()` function, where, during the iteration over each row of the input file, we convert the current amount into USD, specifying the date of the sale to obtain the correct exchange rate. The converted amount is then written to the output file.

- **`make_dim(cs.csv, dim_attributes, output.csv)`**: Used to create the gpu, cpu, and ram tables, as minimal reprocessing was involved, making the creation process more generalizable.

0.2 Assignment 2

In the following Assignment 2, we have been asked to write a second python file that populates the database according to the starting schema in Figure 1.

A step behind this, was the creation of the database schema still referring to Figure 1, using the tool SQL Server Management Studio. We started this process by first constructing the dimensions table, i.e. *Time, Geography, Cpu, Gpu, Ram*, with all the different attributes that characterize each table and setting the primary key for each of them. In general, we used type *int* mainly for primary keys but also for some numeric attributes, *varchar(50)* for the alphanumeric and string attributes and *float* for all those attribute that didn't have an integer value.

Because of a relationship constraints, the last table we created was the fact table *Computer_Sales*. In here, we established that the primary key values would be created automatically by system at any upload/update of the table.

After the creation of all the table, we improved the connectivity of the database schema by setting all the relationship needed to create the foreign key, connecting the fact table with all the dimension tables.

Once the database environment was ready, we loaded all the data prepared in Assignment 1 and we started populating our tables through a Python program: we created a customized function,as follows

```
dataloadCSV(filepath, tablename, connectionstring)
```

that was able to establish a database connection using the proper *connection_string*, read data from a given CSV file into *file_path*, constructs a dynamic parameterized SQL query:

```
placeholders = ', '.join(['?' for  in header])
sqlquery = f"INSERT INTO -tablename (-', '.join(header))
VALUES (-placeholders)"
```

and uses `cursor.executemany(sql_query, data)` command for efficient bulk insertion into the specific table *table_name*, avoiding to upload data row by row. After inserting the data, it commits the changes.

This process was automatically done for each CSV, and for his reason a try-except block was added in order to better handle possible errors during the whole process.

[2]https://pypi.org/project/CurrencyConverter/

Part II - ETL Process

In the second part of this work, we are going to develop and ETL workflow using Visual Studio and the SSIS extension, aiming to answer to the following query: *For each year and region, identify the computer IDs associated with the highest sales of CPUs. Augment the result by including the percentage of sales w.r.t. to the total sales of all computers within the same CPU series.*

The first step was to create the connection to the database in order to extract *time_id* and *year* from the dimension table Time, *geo_id* and *region* from the dimension Geography and *cpu_sales, cpu_id, gpu_id, ram_id, time_id, geo_id* from the fact table ComputerSales. As a second step, we performed a merge between Time and ComputerSales on the key *time_id* and another merge between Geography and the output table of the first merge on the key *geo_id.*

Once we got all the attributes needed to perform our query, we started aggregating attributes grouping on *year, region, cpu_id, gpu_id, ram_id* together with a sum on *cpu_sales* as *total_cpu_sales.*

After these operations, we used a multicast in order to create two different tables on two different flow: on the first table, we performed another aggregation grouping on *cpu_id* and summing on *total_cpu_sales* to obtain *total_sales_cpu_per_cat.* We then merged using as key *cpu_id* this output table with the outgoing flow from the multicast, on which we didn't perform any operation. The operation described so far are shown in the following Figure 2:

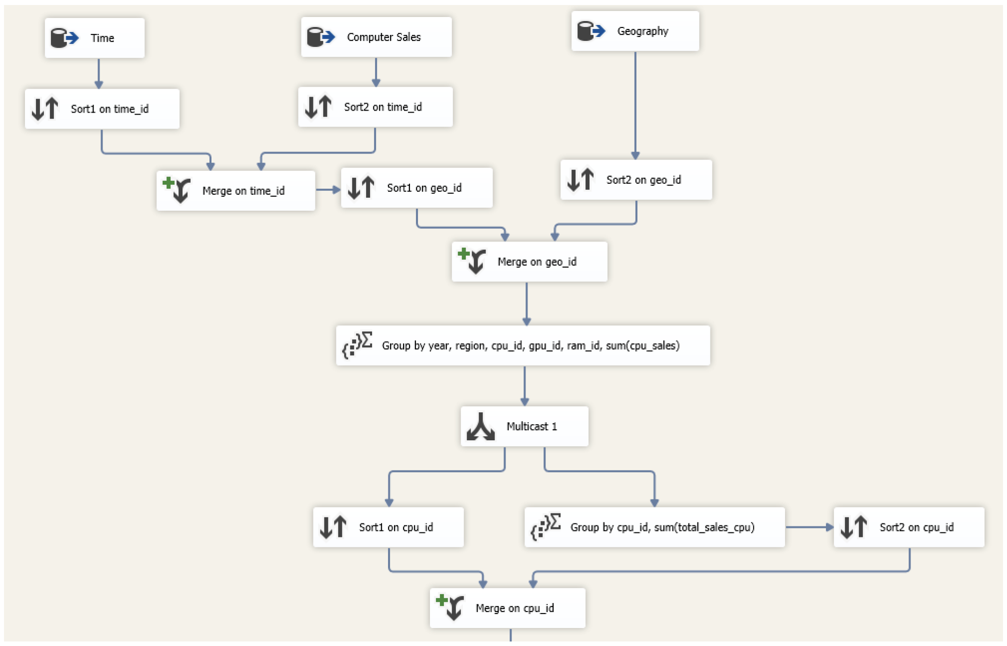


Figura 2: SSIS Data Flow - Part I

At this point, we have all the attribute in order to compute the main interesting part of the query, such

as the ranking in order to find the highest sales of CPUs and the ratio w.r.t to the total sales of all computer with the same CPU. Using a multicast, we divided the workflow into two different part: on one side, we first create the derived column *ratio* dividing the *total_sales_cpu*
total_sales_cpu_per_cat, multiplying the result per 100 to normalize it. On the other side, we performed an aggregation grouping on *year, region* and summing up *total_sales_cpu* in order to retrieve the maximum from the table created in the previous grouping operation as *max_total_sales_cpu*. We then performed a merge using as a key *year, region*, combining the output table of the two flows of the multicast. As last operation, we extracted all the rows matching the conditional split *total_sales_cpu == max_total_sales_cpu*. The second part of the workflow is shown in the Figure 3 as follows:

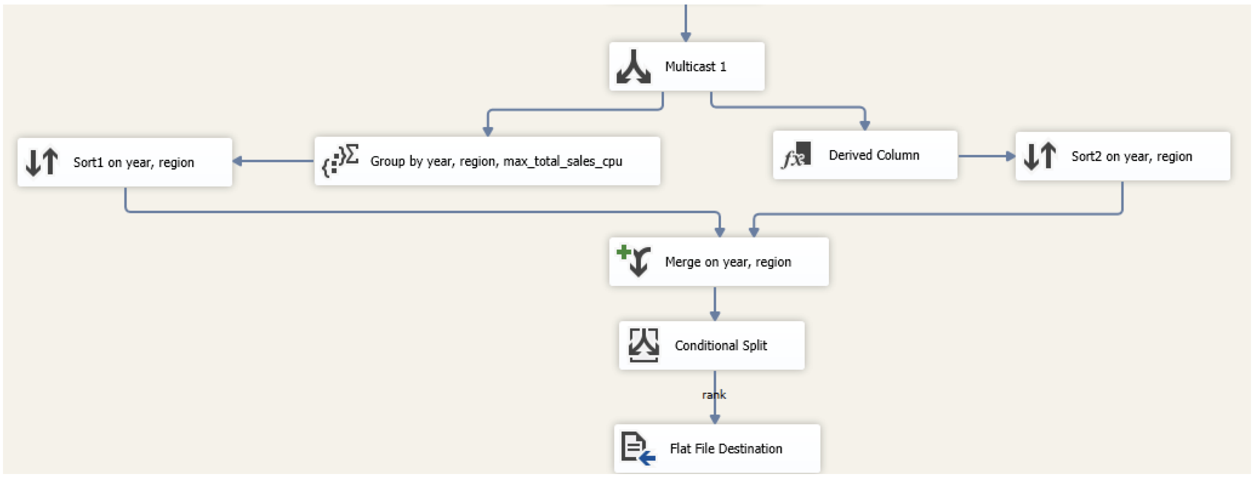


Figura 3: SSIS Data Flow - Part II

Part III - Data Cube (SSAS)

For the creation of the multidimensional cube, we first established the connection with the previously defined database and set up the data views to ensure everything was working correctly. As a preliminary step, we added some useful attributes such as *weekday*, `int` representation of *day_of_week*, and*month_of_year*, literal representation of *month*.

After that, we started creating the dimensions, namely `Geography, Time, Cpu, Gpu, Ram`. Regarding hierarchies, the Ram and Gpu dimensions were kept flat, while for the others, we identified some multilevel hierarchies:

- **Geography:** The hierarchy here is of type *Region* $\rightarrow$ *Country* $\rightarrow$ *Continent*.
- **Time:** The hierarchy here is of type *DayOfWeek* $\rightarrow$ *Week* $\rightarrow$ *MonthOfYear* $\rightarrow$ *Quarter* $\rightarrow$ Year. Additionally, we established functional dependencies of type *DayOfWeek* $\rightarrow$ *Weekday* and *MonthOfYear* $\rightarrow$ *Month* to sort the literal attributes based on their numerical order, as expressed by their `int` representation.
- **Cpu:** The hierarchies here are of types *CpuSeries* $\rightarrow$ *CpuBrand* and *CpuName* $\rightarrow$ *CpuSeries*.

After creating the hierarchies, we selected `computer_sales` as the fact table and chose the *..._sales* and *..._sales_usd* fields as measures, with "sum" as the aggregation function. We also included a count for each sale.

Finally, we deployed the cube to the corresponding server using the HTTP protocol, with the final schema looking as in figure 4

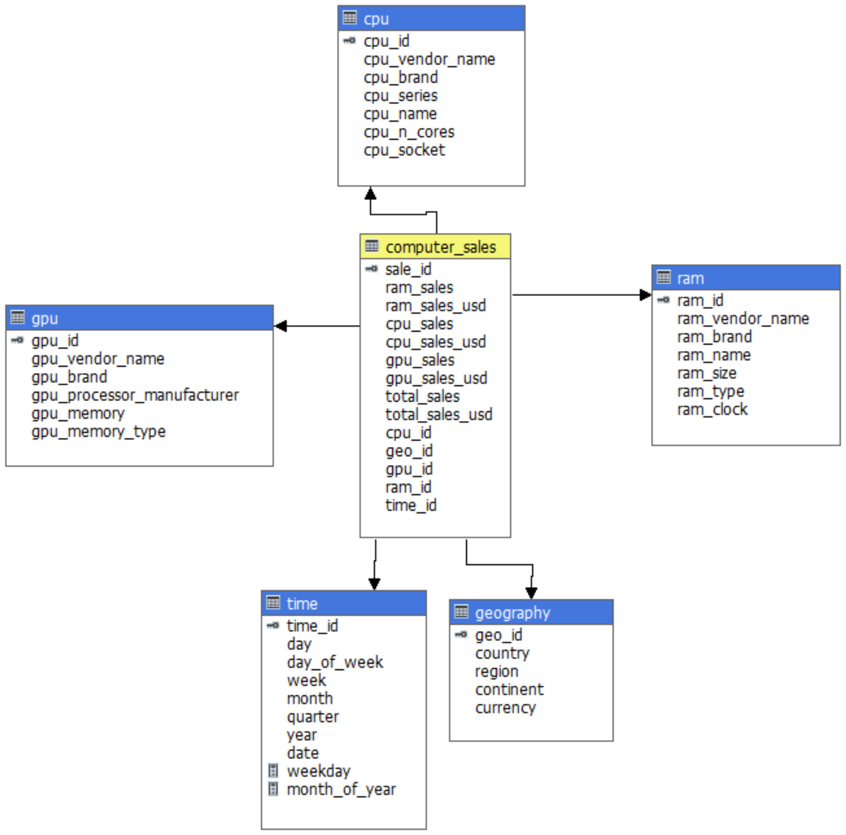


Figura 4: Data view from Visual Studio

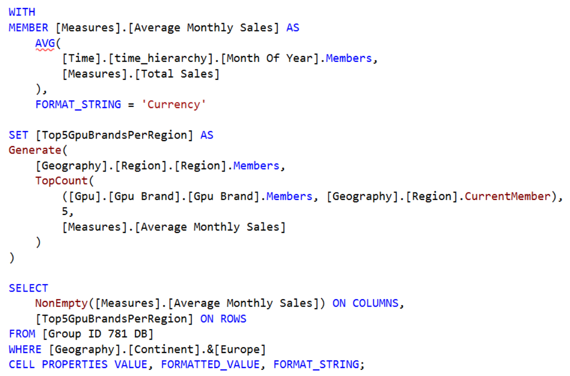
Part IV - MDX Query

The goal was to show the top 5 cpu, ram, and gpu brands w.r.t the monthly average sales for each region in Europe. To better answer this question, we decided to split the report into 3 smaller ones, providing a more concise representation. The approach used is identical for all three categories, with minor changes for cpu to account for the smaller numerosity of the available brands.

For what regards Gpu, the first thing was to define a new member, namely Average Monthly Sales, which we used as the main measure to sort our brands by. This new measure was computed by means of the predefined AVG() function, passing the set of months and the Total Sales as numeric expression. Note that we used MonthOfYear from the hierarchy, such that we could average on each month of each year, instead of the aggregation of months.

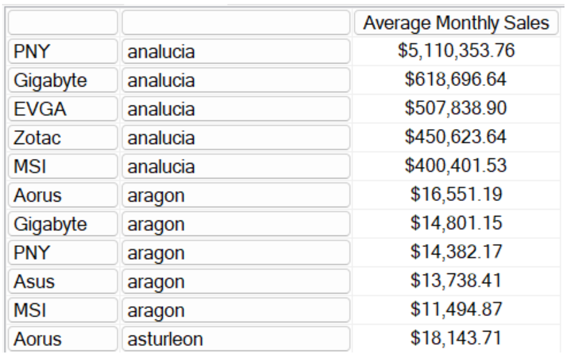
We then proceeded to compute the main result of the report, called Top5GpuBrandsPerRegion. This was done by means of the GENERATE() function. We passed all the regions to the function to loop on and in the TOPCOUNT() function specified the tuples to consider for each region (Gpu brand in our case with the current Region), set the count to 5 and the previously defined Average Monthly Sales as sorting expression.

As a last step, we selected the resulting tuples on the ROWS with the measure on the COLUMNS, formatted as currency. Of course we also specified in WHERE clause that we're interested only in the European regions. The query and results for Gpu are shown in figure 5 and figure 6



```
WITH
MEMBER [Measures].[Average Monthly Sales] AS
    AVG(
        [Time].[time_hierarchy].[Month Of Year].Members,
        [Measures].[Total Sales]
    ),
    FORMAT_STRING = 'Currency'

SET [Top5GpuBrandsPerRegion] AS
Generate(
    [Geography].[Region].[Region].Members,
    TopCount(
        ([Gpu].[Gpu Brand].[Gpu Brand].Members, [Geography].[Region].CurrentMember),
        5,
        [Measures].[Average Monthly Sales]
    )
)

SELECT
    NonEmpty([Measures].[Average Monthly Sales]) ON COLUMNS,
    [Top5GpuBrandsPerRegion] ON ROWS
FROM [Group ID 781 DB]
WHERE [Geography].[Continent].&[Europe]
CELL PROPERTIES VALUE, FORMATTED_VALUE, FORMAT_STRING;
```

Figura 5: MDX query for Gpu

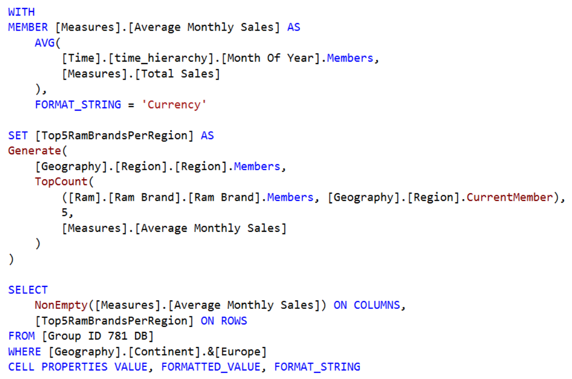


		Average Monthly Sales
PNY	analucia	$5,110,353.76
Gigabyte	analucia	$618,696.64
EVGA	analucia	$507,838.90
Zotac	analucia	$450,623.64
MSI	analucia	$400,401.53
Aorus	aragon	$16,551.19
Gigabyte	aragon	$14,801.15
PNY	aragon	$14,382.17
Asus	aragon	$13,738.41
MSI	aragon	$11,494.87
Aorus	asturleon	$18,143.71

Figura 6: Results for Gpu

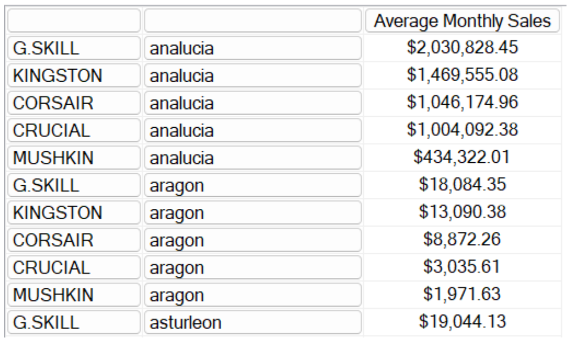
The same reasoning was followed for Ram as well, as shown in figure 7 and figure 8

In the case of Cpu, we set TOPCOUNT() to 1 since only 2 brands are available, making the original report useless as most of values would be null. Figure 9 and 10 for reference.



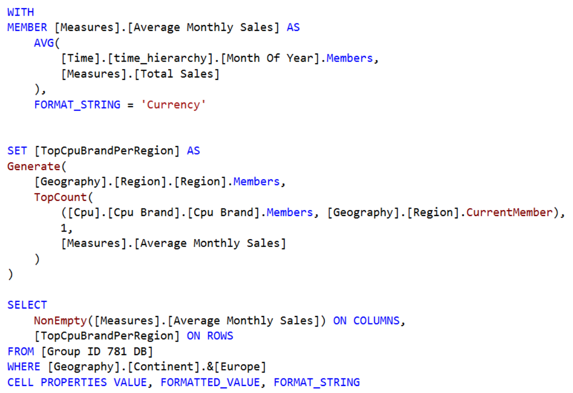
```
WITH
MEMBER [Measures].[Average Monthly Sales] AS
    AVG(
        [Time].[time_hierarchy].[Month Of Year].Members,
        [Measures].[Total Sales]
    ),
    FORMAT_STRING = 'Currency'

SET [Top5RamBrandsPerRegion] AS
Generate(
    [Geography].[Region].[Region].Members,
    TopCount(
        ([Ram].[Ram Brand].[Ram Brand].Members, [Geography].[Region].CurrentMember),
        5,
        [Measures].[Average Monthly Sales]
    )
)

SELECT
    NonEmpty([Measures].[Average Monthly Sales]) ON COLUMNS,
    [Top5RamBrandsPerRegion] ON ROWS
FROM [Group ID 781 DB]
WHERE [Geography].[Continent].&[Europe]
CELL PROPERTIES VALUE, FORMATTED_VALUE, FORMAT_STRING
```

Figura 7: MDX query for Ram



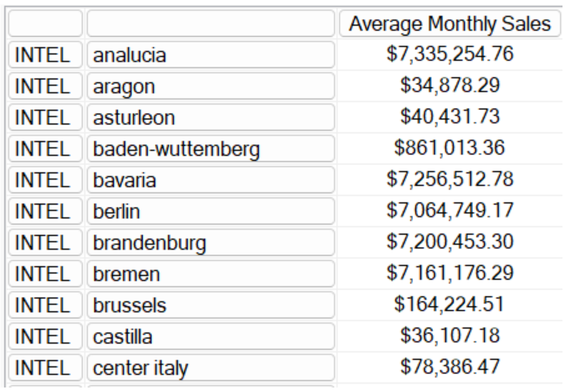
		Average Monthly Sales
G.SKILL	analucia	$2,030,828.45
KINGSTON	analucia	$1,469,555.08
CORSAIR	analucia	$1,046,174.96
CRUCIAL	analucia	$1,004,092.38
MUSHKIN	analucia	$434,322.01
G.SKILL	aragon	$18,084.35
KINGSTON	aragon	$13,090.38
CORSAIR	aragon	$8,872.26
CRUCIAL	aragon	$3,035.61
MUSHKIN	aragon	$1,971.63
G.SKILL	asturleon	$19,044.13

Figura 8: Results for Ram



```
WITH
MEMBER [Measures].[Average Monthly Sales] AS
    AVG(
        [Time].[time_hierarchy].[Month Of Year].Members,
        [Measures].[Total Sales]
    ),
    FORMAT_STRING = 'Currency'


SET [TopCpuBrandPerRegion] AS
Generate(
    [Geography].[Region].[Region].Members,
    TopCount(
        ([Cpu].[Cpu Brand].[Cpu Brand].Members, [Geography].[Region].CurrentMember),
        1,
        [Measures].[Average Monthly Sales]
    )
)

SELECT
    NonEmpty([Measures].[Average Monthly Sales]) ON COLUMNS,
    [TopCpuBrandPerRegion] ON ROWS
FROM [Group ID 781 DB]
WHERE [Geography].[Continent].&[Europe]
CELL PROPERTIES VALUE, FORMATTED_VALUE, FORMAT_STRING
```

Figura 9: MDX query for Cpu



		Average Monthly Sales
INTEL	analucia	$7,335,254.76
INTEL	aragon	$34,878.29
INTEL	asturleon	$40,431.73
INTEL	baden-wuttemberg	$861,013.36
INTEL	bavaria	$7,256,512.78
INTEL	berlin	$7,064,749.17
INTEL	brandenburg	$7,200,453.30
INTEL	bremen	$7,161,176.29
INTEL	brussels	$164,224.51
INTEL	castilla	$36,107.18
INTEL	center italy	$78,386.47

Figura 10: Results for Cpu

Part V - Dashboards

In the final part of our project, i.e. Assignment 6, we have been asked to create a plot/dashboard of our choice that we deem interesting w.r.t. the data available in our cube. We completed this task using PowerBI and establishing a connection to the cube we created in the previous steps. The following figures aim to point out some interesting aspects and insight from our data.

First, we decided to use a filled map to represent *computer_sales_count* and *total_sales* at different granularities, following the hierarchy of *continent*, *country*, and *region*. This allows us to visualize the geographical distribution of sales. For clearer and more accessible data presentation, we also created a stacked bar chart. This chart shows the countries within each continent, with the size of each bar proportional to the sales volume. As shown in Figure 11, in America and Oceania, sales are mainly split between the United States, Canada, and Australia, New Zealand, respectively. In contrast, Germany dominates the sales scene in Europe.

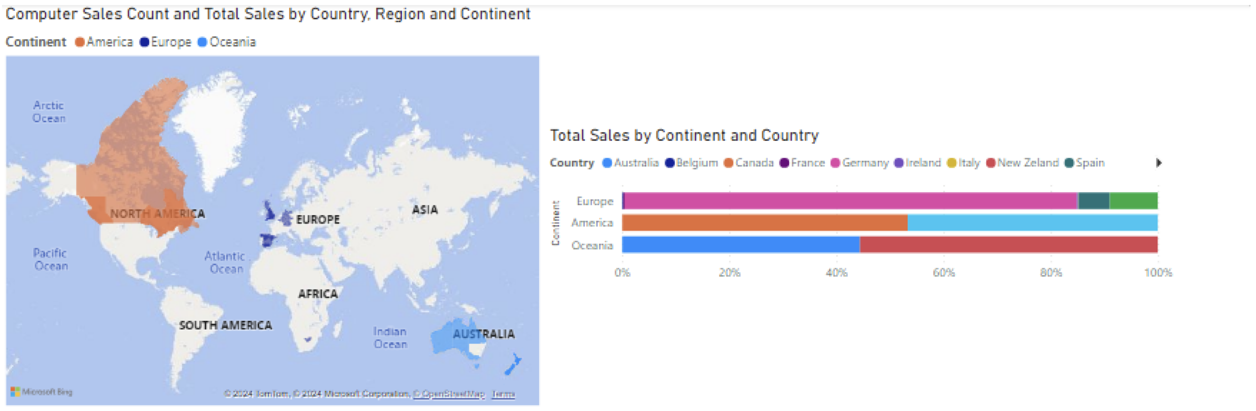


Figura 11: Computer_Sales and Total_Sales by Continent, Country and Region

Another interesting aspect is shown in Figure 12, which displays the annual sales of CPUs, GPUs, and RAM using box plots. Additionally, a line graph illustrates the overall trend of *total_sales* over the years. From this plot, we can see that CPU sales are generally much higher than RAM sales. Notably, 2017 was the most profitable year, with a nearly linear growth trend leading up to it. However, in 2018, there was a significant drop, with sales for all components halving compared to the previous year.

As a final analysis, we decided to use the pie charts in Figure 13 to show the total number of computers sold, categorized by their CPU, GPU, and RAM brands. This helped us see which brands were most popular in the market. The chart clearly showed that INTEL was the top choice for CPUs. For GPUs, PNY and GigaByte, and for RAM, GSKILL and KINGSTONE were the leading brands. Other brands also had a significant presence in the market, but these were the most prominent ones. This visualization gave us a good understanding of consumer preferences for different computer component brands, helping us identify the most popular choices.

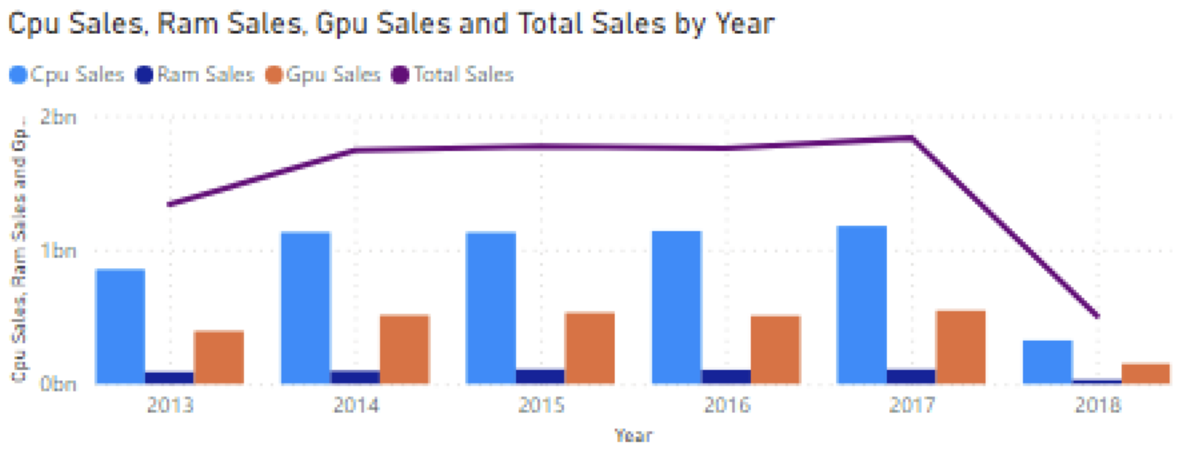


Figura 12: Cpu_Sales, Ram_Sales, Gpu_Sales and Total_Sales by Year

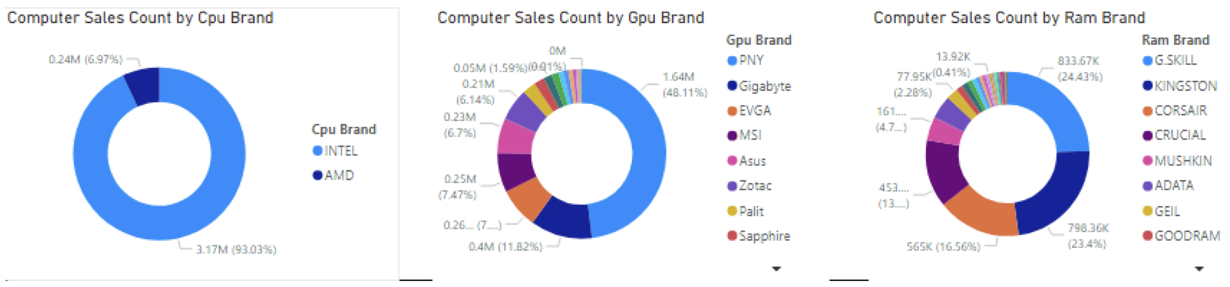


Figura 13: Computer_Sales_Count by Cpu_Brand, Gpu_Brand and Ram_Brand