Decision Support Systems
Laboratory of Data Science Project

Simone Di Luna

Department of Computer Science
University of Pisa

July 2023

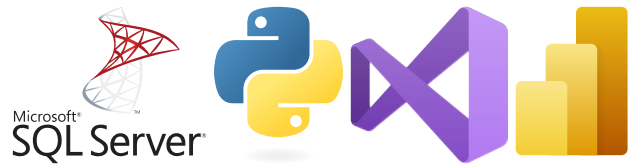
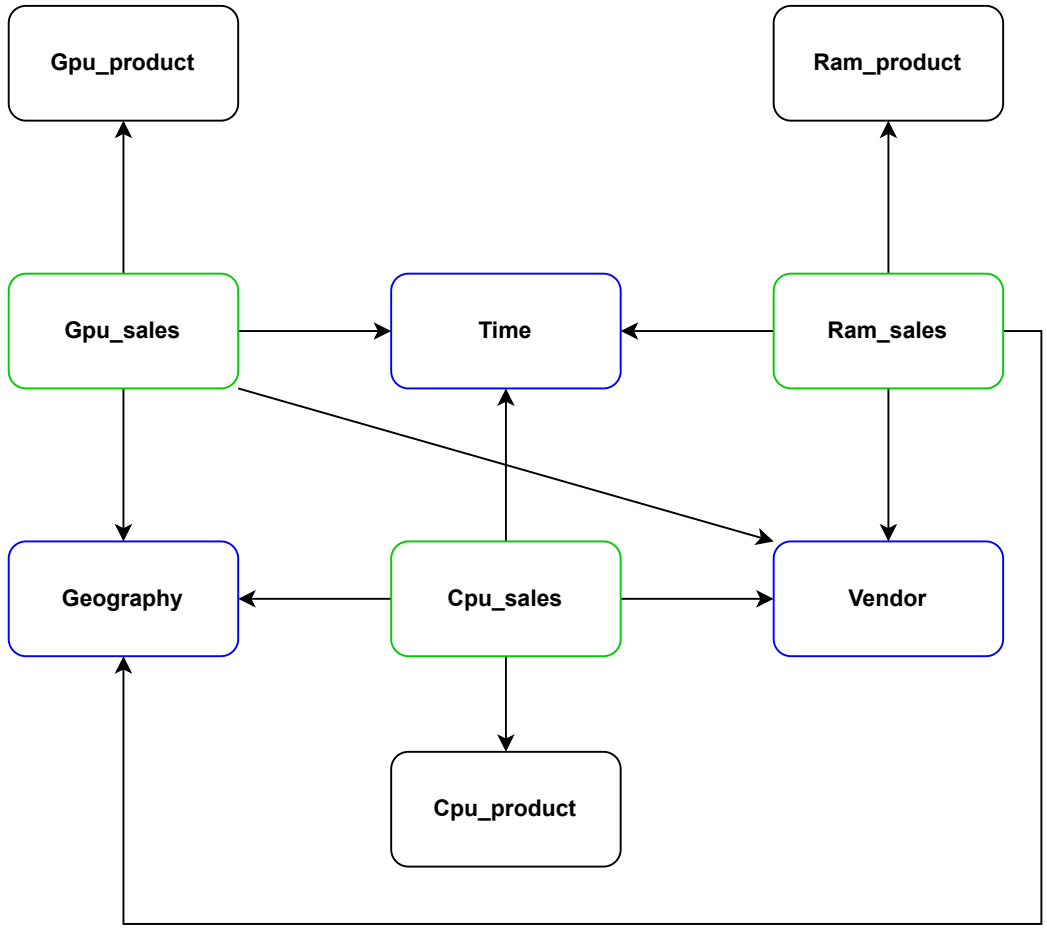


Table of Contents

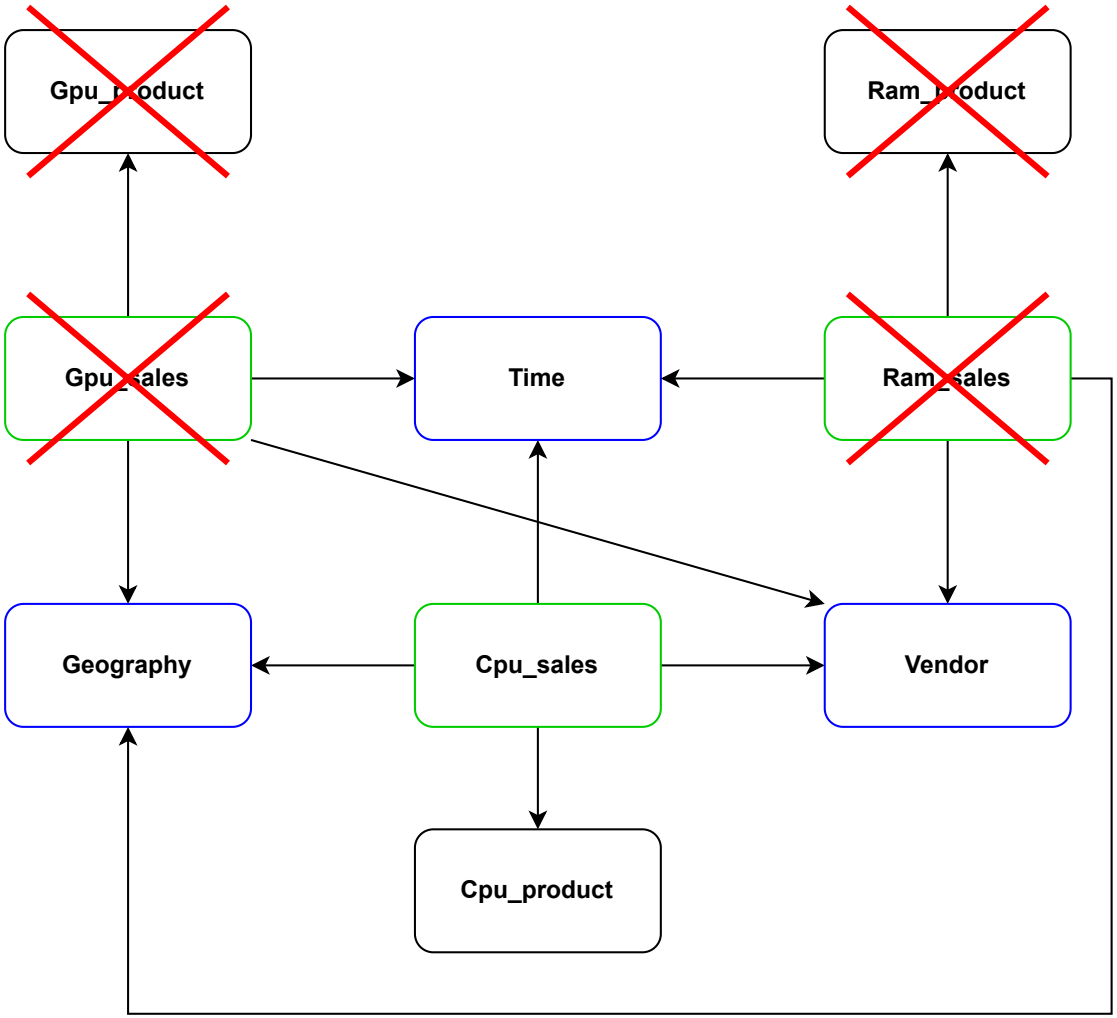
From the data warehouse...

- Fact tables
- Conformed dimension tables
- Simple dimension tables
- Referential constraints (arrows)



Constellation schema

. . . to the CPU data mart



Star schema

Composition of CSV files

fact.csv

- Id
- gpu_code
- cpu_code
- ram_code
- time_code
- geo_code
- vendor_code
- sales_uds
- sales_currency

cpu.csv

- cpu_code
- brand
- series
- name
- n_cores
- socket

geography.csv

- geo_code
- continent
- country
- region
- currency

time.csv

- time_code
- year
- month
- day
- week

vendor.csv

- vendor_code
- name

Preliminary analysis

For each CSV file and for each variable in the file, we are interested to discover the following information:

- Data type.
- Domain.
- Character encoding (if qualitative).
- Number of missing values and form in which they appear.

Why?

Preliminary analysis

For each CSV file and for each variable in the file, we are interested to discover the following information:

- Data type.
- Domain.
- Character encoding (if qualitative).
- Number of missing values and form in which they appear.

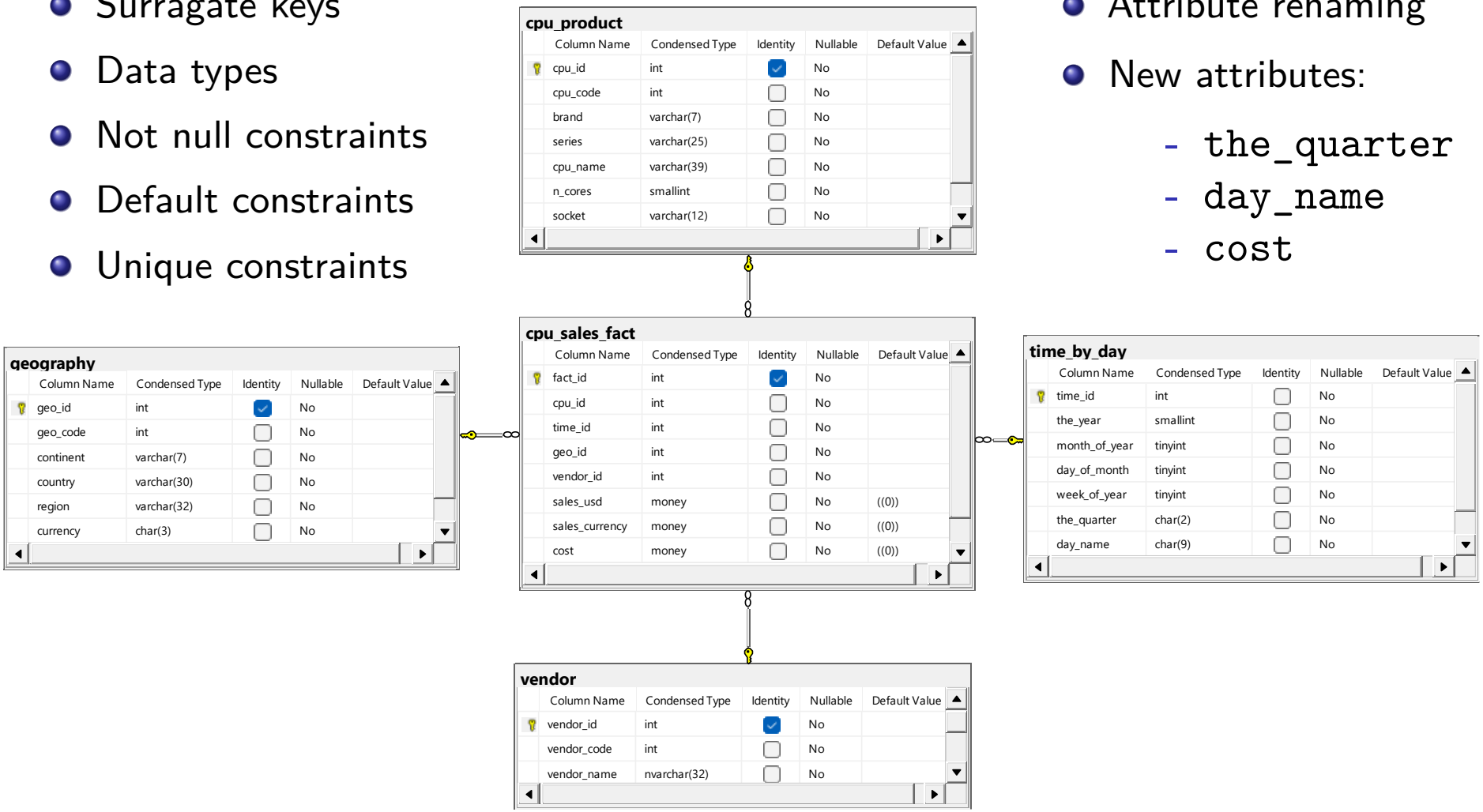
Why?

Preliminary analysis

- To help us choose between different data types and arguments in MS SQL Server:
  - `char`/`varchar` or `nchar`/`nvarchar`?
  - What about the length?
  - `int`, `smallint`, `tinyint`, `money`,...?
- To check the semantic accuracy.

simonediluna_DB: Diagram

- Surragate keys
- Data types
- Not null constraints
- Default constraints
- Unique constraints



- Attribute renaming
- New attributes:
  - `the_quarter`
  - `day_name`
  - `cost`

simonediluna_DB: Indexes

Before populating the database

Clustered indexes on primary keys (auto-generated).

After populating the database

- Non-clustered indexes on foreign keys of the fact table.
- Non-clustered columnar indexes on the attributes that was assumed to be used most frequently in the `WHERE`/`GROUP BY` clauses (1 index for each dimension table).

simonediluna_DB: Indexes

Before populating the database

Clustered indexes on primary keys (auto-generated).

After populating the database

- Non-clustered indexes on foreign keys of the fact table.
- Non-clustered columnar indexes on the attributes that was assumed to be used most frequently in the `WHERE`/`GROUP BY` clauses (1 index for each dimension table).

Missing values

Which attributes are affected?

Attribute	File	Type	Number
`series`	cpu	"*Unknown*"	1
`cpu_code`	fact	""	4 409 487
`gpu_code`	fact	""	5 017 522
`ram_code`	fact	""	2 602 347

Missing values

How to treat them?

Missing values in `fact.csv`

Skip the rows where `cpu_code = ""` and remove the `gpu_code` and `ram_code` columns.

Missing value in `cpu.csv`

Infer the series of the CPU by looking at its name:

cpu_code	brand	series	name	n_cores	socket
467	AMD	Unknown	Amd E-Series E2-3200	2	FM1

Missing values

How to treat them?

Missing values in `fact.csv`

Skip the rows where `cpu_code` = "" and remove the `gpu_code` and `ram_code` columns.

Missing value in `cpu.csv`

Infer the series of the CPU by looking at its name:

cpu_code	brand	series	name	n_cores	socket
467	AMD	Unknown	Amd E-Series E2-3200	2	FM1

Missing values

How to treat them?

Missing values in `fact.csv`

Skip the rows where `cpu_code = ""` and remove the `gpu_code` and `ram_code` columns.

Missing value in `cpu.csv`

Infer the series of the CPU by looking at its name:

cpu_code	brand	series	name	n_cores	socket
467	AMD	Unknown	Amd E-Series E2-3200	2	FM1

Other inconsistencies in the cpu.csv file

cpu_code	brand	series	name	n_cores	socket
1	AMD	Amd 2650	Amd Sempron 2650	2	AMD Socket AM1
2	AMD	Amd 3850	Amd Sempron 3850	4	AMD Socket AM1
3	AMD	Amd 5150	Amd Athlon 5150	4	AMD Socket AM1
4	AMD	Amd 5350	Amd Athlon 5350	4	AMD Socket AM1
187	AMD	Amd E2-3200	Amd E2-3200	2	AMD Socket FM1
200	AMD	Amd E-Series E-450	Amd E-Series E-450	2	Intel Socket BGA413
240	AMD	Amd Opteron	Amd Opteron 2373 Ee	4	Intel Socket LGA3647
467	AMD	Unknown	Amd E-Series E2-3200	2	AMD Socket FM1

Domain knowledge

Other inconsistencies in the `cpu.csv` file

How to solve them?

- Infer the correct value from the other fields, if possible.
- For the socket field remove redundant information (*i.e.*, brand + *"Socket"*).
- Look for information on the manufacturer's official website.

Other inconsistencies in the `cpu.csv` file

How to solve them?

- Infer the correct value from the other fields, if possible.
- For the `socket` field remove redundant information (*i.e.*, brand + *"Socket"*).
- Look for information on the manufacturer's official website.

Other inconsistencies in the `cpu.csv` file

How to solve them?

- Infer the correct value from the other fields, if possible.
- For the `socket` field remove redundant information (*i.e.*, brand + *"Socket"*).
- Look for information on the manufacturer's official website.

Other inconsistencies in the `cpu.csv` file

How to solve them?

- Infer the correct value from the other fields, if possible.
- For the `socket` field remove redundant information (*i.e.*, brand + *"Socket"*).
- Look for information on the manufacturer's official website.

Computation of new attributes: Quarters

The quarter in which a fact occurred can be easily inferred from the month of the year:

```
def compute_quarter(month: str):
    return ["Q1", "Q2", "Q3", "Q4"][(int(month)-1) // 3]
```

Computation of new attributes: Name of the day of the week

The name of the day of the week on which a particular fact occurred can be inferred from the `time_id`. Specifically, it is necessary to fix a point in time where the name of the day is known and then count the days elapsed from there to the date of the fact.

What time reference point to choose?

The Unix epoch: Thursday, January 1, 1970.
The earliest date in the `time` dimension table is March 22, 2013.

Computation of new attributes: Name of the day of the week

The name of the day of the week on which a particular fact occurred can be inferred from the `time_id`. Specifically, it is necessary to fix a point in time where the name of the day is known and then count the days elapsed from there to the date of the fact.

What time reference point to choose?

The Unix epoch: Thursday, January 1, 1970.
The earliest date in the `time` dimension table is March 22, 2013.

Computation of new attributes: Name of the day of the week

The name of the day of the week on which a particular fact occurred can be inferred from the `time_id`. Specifically, it is necessary to fix a point in time where the name of the day is known and then count the days elapsed from there to the date of the fact.

What time reference point to choose?

The Unix epoch: Thursday, January 1, 1970.
The earliest date in the `time` dimension table is March 22, 2013.

Computation of new attributes: Cost

$C = \frac{R}{k+1}$, where $k = \frac{R-C}{C}$ is the markup which was randomly drawn from a uniform distribution.

```
def compute_cost(
        revenue: str,
        markup_range=(0.15, 0.4)
    ) -> float:
    rnd = random.random()
    delta = markup_range[1] - markup_range[0]
    markup = rnd * delta + markup_range[0]  # min-max norm.
    return float(revenue) / (1 + markup)  # C
```

Data warehouse population

The chronological steps:

1. Populate the dimension tables.
2. For each dimension table in the DB, compute a dictionary mapping the values of the old PK to the auto-generated values of the surrogate key.
3. Replace the old FKs in the `fact.csv` file with the new FKs referencing the auto-generated PKs.
4. Populate the fact table.

Data warehouse population

The chronological steps:

1. Populate the dimension tables.
2. For each dimension table in the DB, compute a dictionary mapping the values of the old PK to the auto-generated values of the surrogate key.
3. Replace the old FKs in the `fact.csv` file with the new FKs referencing the auto-generated PKs.
4. Populate the fact table.

Data warehouse population

The chronological steps:

1. Populate the dimension tables.
2. For each dimension table in the DB, compute a dictionary mapping the values of the old PK to the auto-generated values of the surrogate key.
3. Replace the old FKs in the `fact.csv` file with the new FKs referencing the auto-generated PKs.
4. Populate the fact table.

Data warehouse population

The chronological steps:

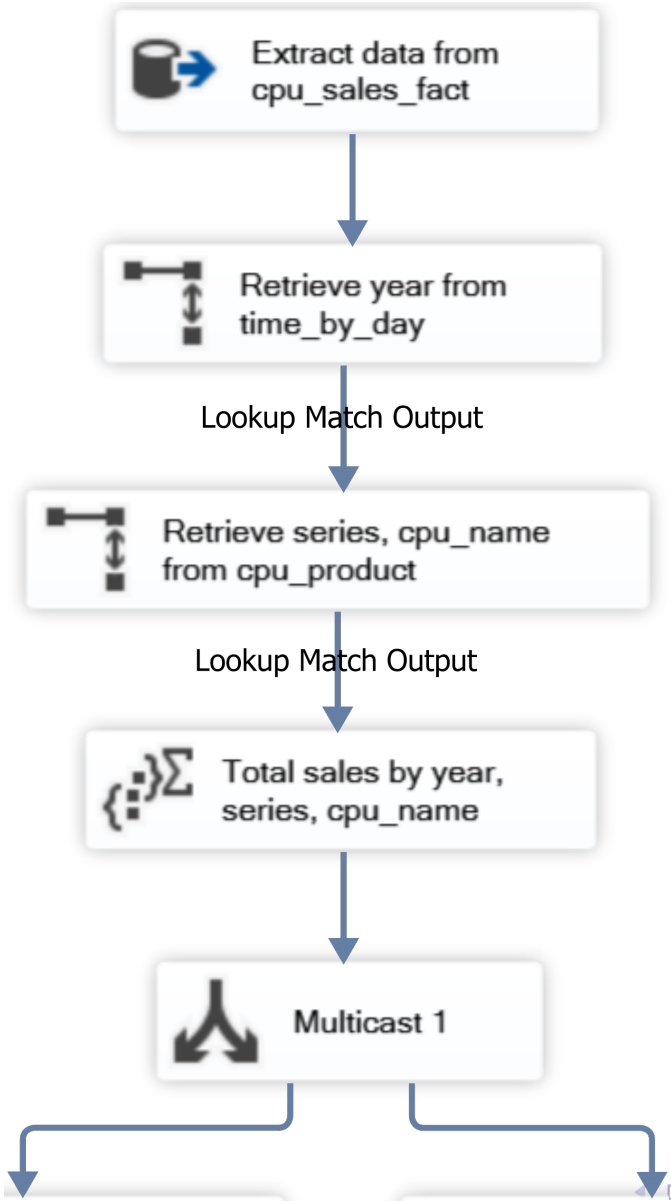
1. Populate the dimension tables.
2. For each dimension table in the DB, compute a dictionary mapping the values of the old PK to the auto-generated values of the surrogate key.
3. Replace the old FKs in the `fact.csv` file with the new FKs referencing the auto-generated PKs.
4. Populate the fact table.

ETL task

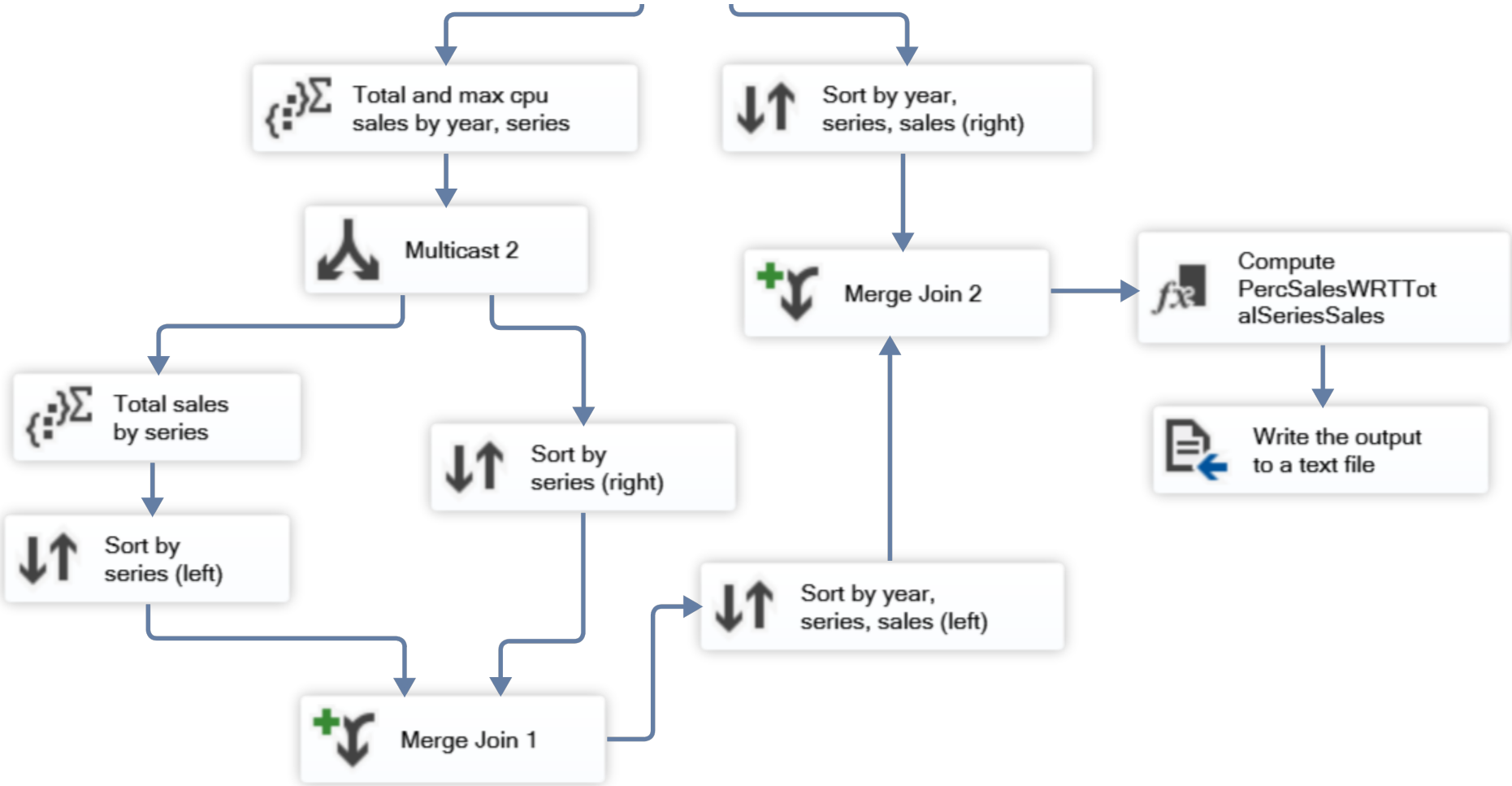
ETL workflow in Visual Studio

For each year and CPU series, the name of the CPU that sold the most that year and the percentage in sales with respect to the total sales of the corresponding CPU series.

SSIS solution part 1



SSIS solution part 2



Preliminary settings

Server:	`http://lds.di.unipi.it/olap/msmdpump.dll`
Database:	`simonediluna_SSAS_DB`
Conn. string:	`Provider=SQLOLEDB.1;`
	`Data Source=lds.di.unipi.it;`
	`Password=**********;`
	`User ID=simonediluna;`
	`Initial Catalog=simonediluna_DB`

Dimensions

The cube has 4 dimensions, each based solely on one table in the data warehouse:

- Time
- Vendor
- CPU
- Geography

Measures

Two group of measures:

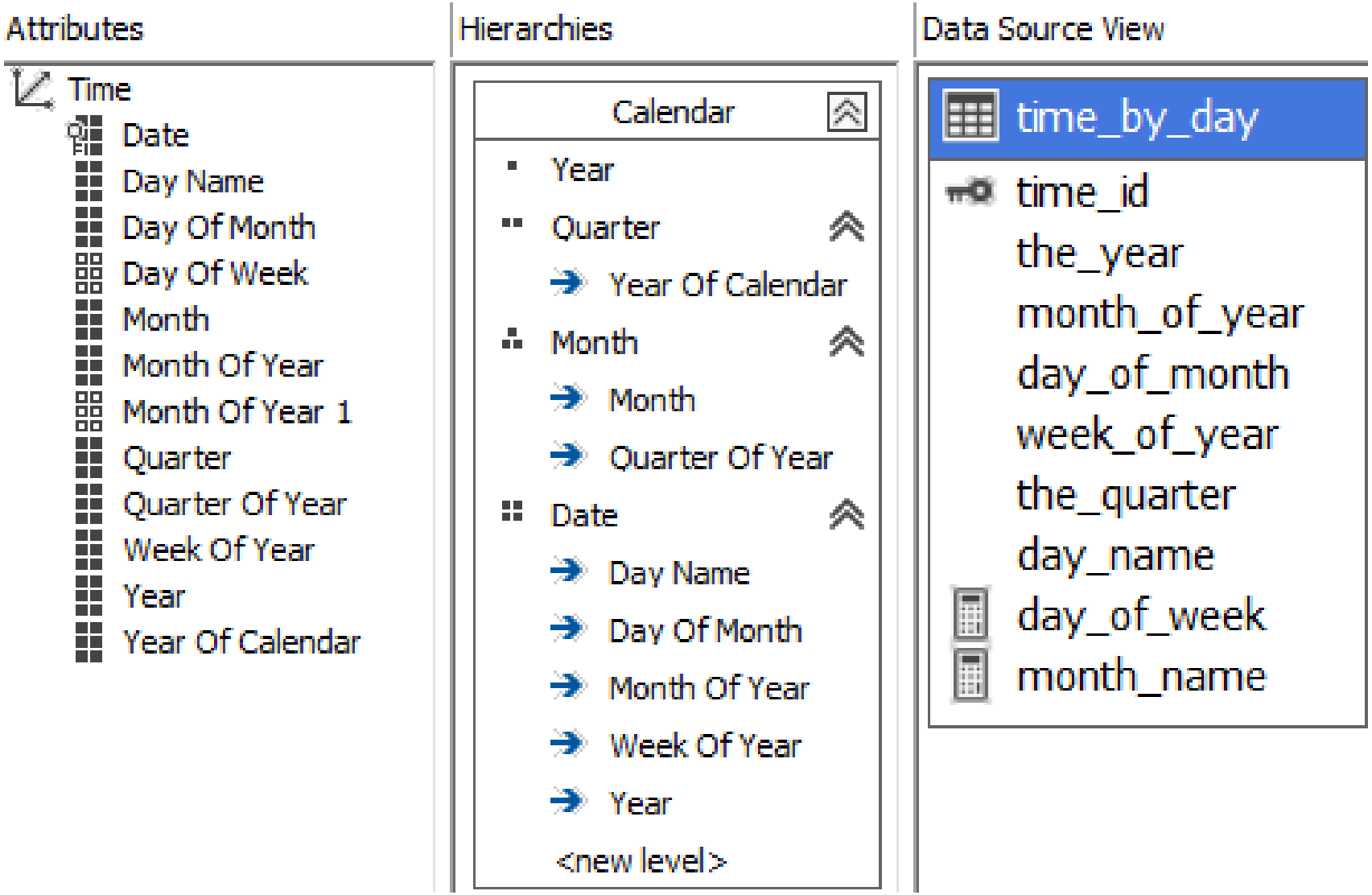
CPU Sales Fact

- Sales Usd (`Sum, Currency`)
- Sales Currency (`Sum, Currency`)
- Cost (`Sum, Currency`)
- Cpu Sales Fact Count (`Count, Standard`)
  - Gross Profit (`Currency`)
  - Margin (`Percent`)
  - Markup (`Percent`)

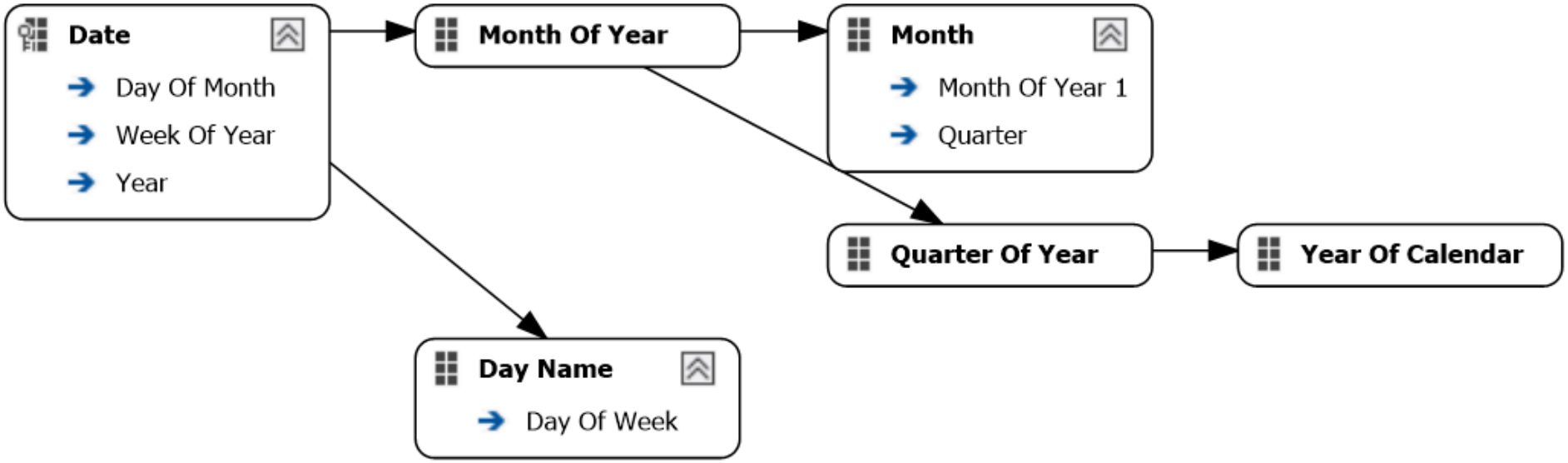
Vendor

- NVendors (`DistinctCount, Standard`)

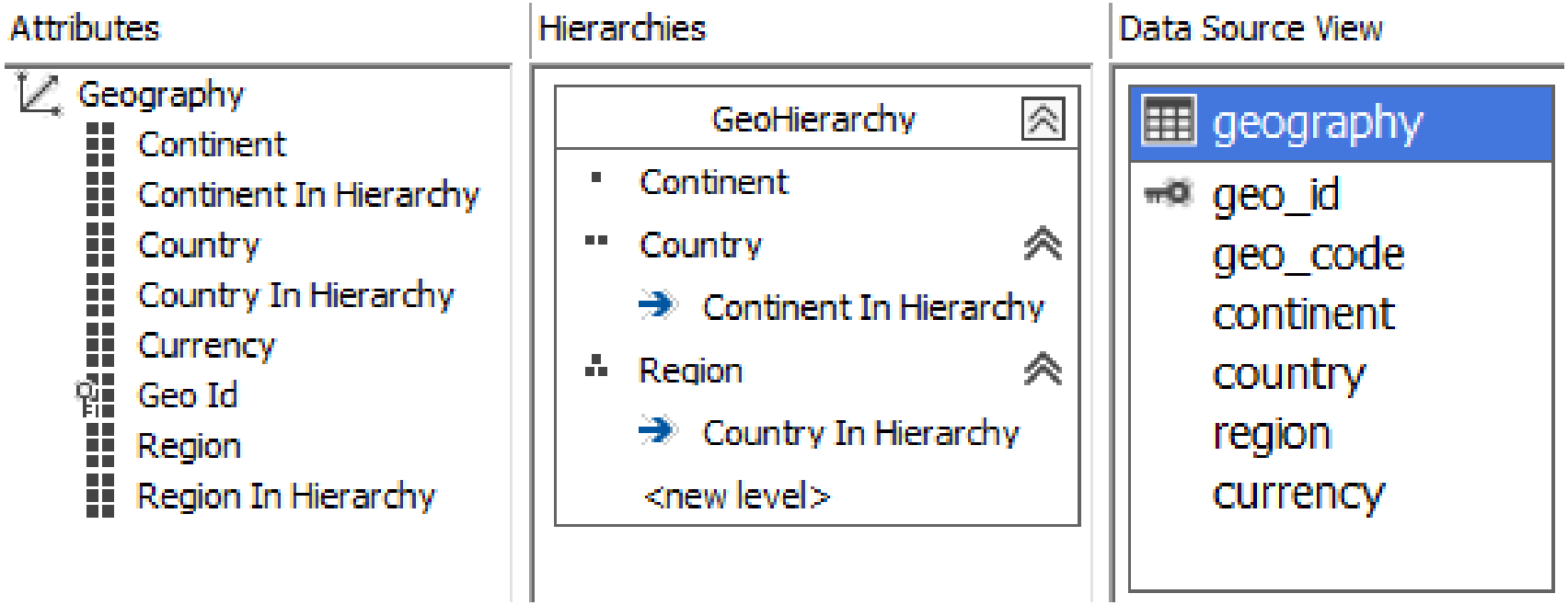
Time dimension: Structure

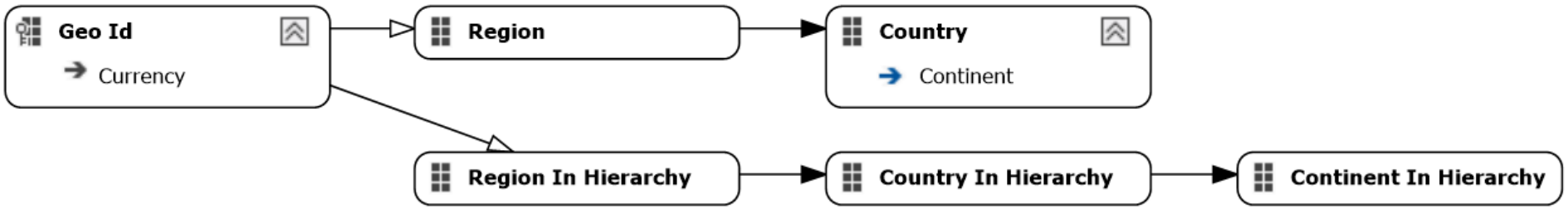


Time dimension: Attribute Relationships

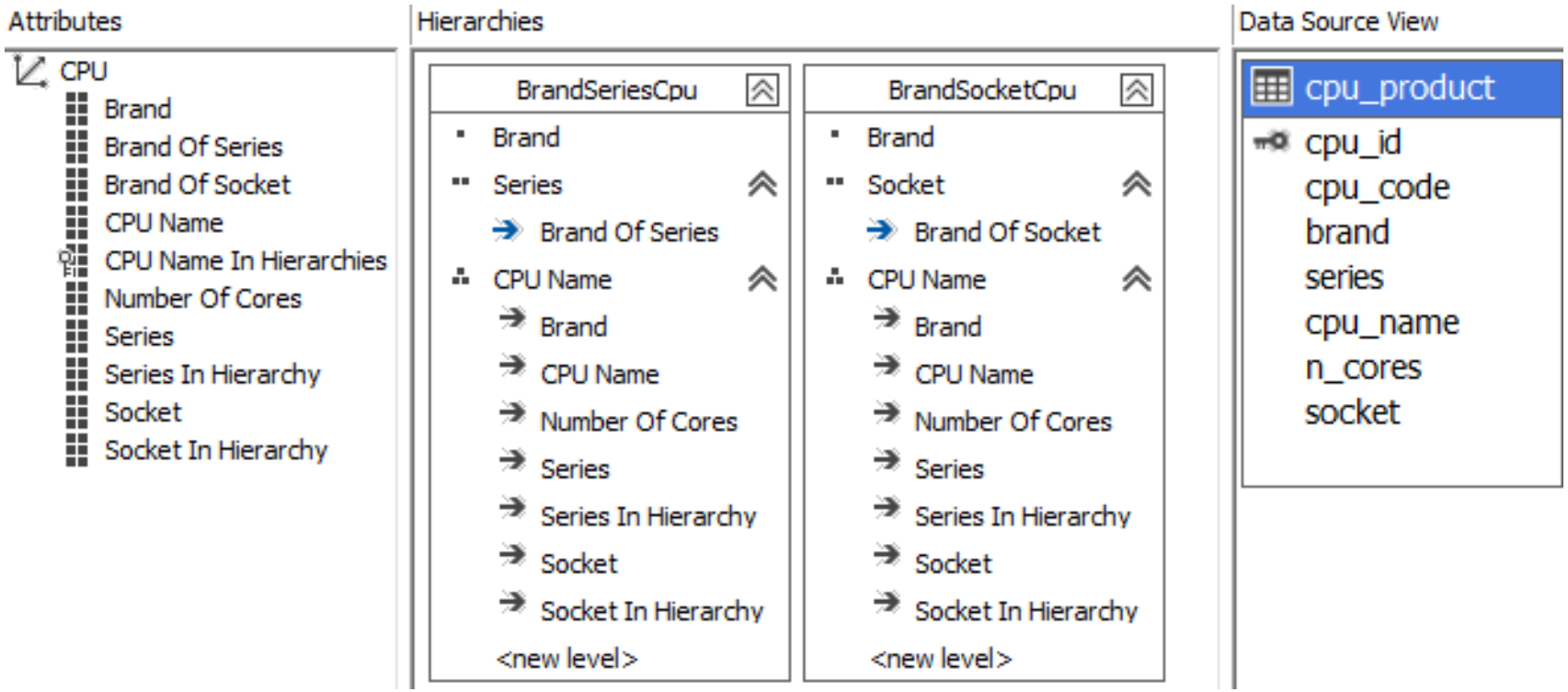


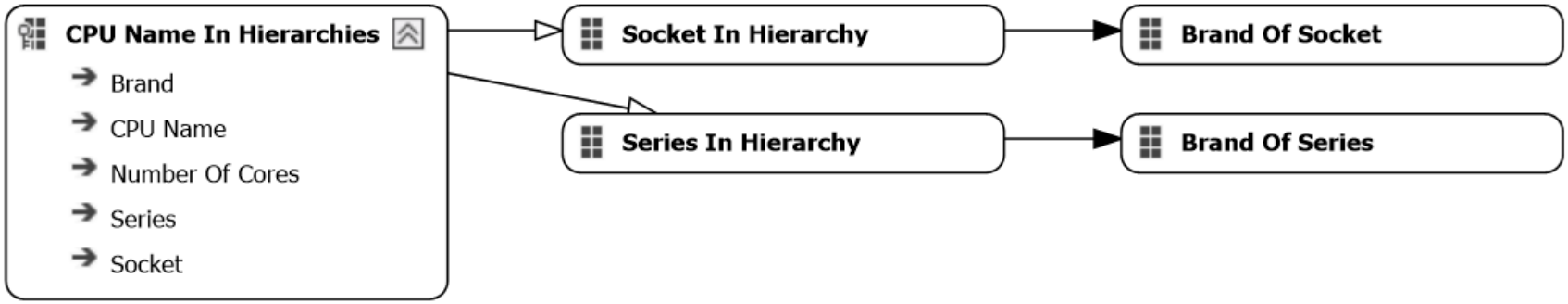
Geography dimension





CPU dimension





Storage mode

The default storage mode was not modified: pure MOLAP without proactive caching.

MDX query

Analysis to perform with MDX

For each CPU brand, the percentage variation of profit (sales - cost) with respect to the previous year, for each year.

Resolution method 1: PREVMEMBER

```
WITH MEMBER [% Profit Variation] AS
   ([Profit] - ([Time].[Year].PREVMEMBER, [Profit]))
      / ([Time].[Year].PREVMEMBER, [Profit]),
   FORMAT_STRING = 'Percent'
SELECT
   [% Profit Variation] ON 0,
   ([CPU].[Brand].[Brand], [Time].[Year].[Year]) ON 1
FROM [CPU Sales];
```

Resolution method 2: LAG

```
WITH MEMBER [% Profit Variation] AS
   ([Profit] - ([Time].[Year].CURRENTMEMBER.LAG(1), [Profit]))
      / ([Time].[Year].CURRENTMEMBER.LAG(1), [Profit]),
   FORMAT_STRING = 'Percent'
SELECT
   [% Profit Variation] ON 0,
   ([CPU].[Brand].[Brand], [Time].[Year].[Year]) ON 1
FROM [CPU Sales];
```

Resolution method 3: PARALLELPERIOD

```
WITH
   MEMBER [Profit Prev Year] AS
      (
         PARALLELPERIOD(
            [Time].[Calendar].[Year],
            1,
            [Time].[Calendar].CURRENTMEMBER
         ),
         [Profit]
      )
   MEMBER [% Profit Variation] AS
      ([Profit] - [Profit Prev Year]) / [Profit Prev Year],
      FORMAT_STRING = 'Percent'
SELECT
   [% Profit Variation] ON 0,
   ([CPU].[Brand].[Brand], [Time].[Calendar].[Year]) ON 1
FROM [CPU Sales];
```

Power BI dashboard