University of Pisa

Master's degree in

Data Science & Business Informatics

Laboratory of Data Science

Project Assignment - Part 1

Professors:

Anna Monreale, Roberto Pellungrini

Group29:

Alfonso Ferraro

Student ID: 626006

E-mail: a.ferraro16@studenti.unipi.it

Michele Andreucci

Student ID: 628505

E-mail: m.andreucci4@studenti.unipi.it

Introduction

With given datasets, the aim of this report is to describe all the processes that will lead to the creation of a database and its data management/preprocessing in ***Chapter 1***. After that, some problems in the created database will be addressed through SSIS (SQL Server Integration Services) in ***Chapter 2***. In the end, in ***Chapter 3***, there will be the construction of a *Datacube*, in which some problems will be addressed through MDX (MultiDimensional eXpressions) queries, and data visualization through dashboards created on Power BI.

Chapter 1

Creation of the database and data management

The datasets we'll be working with are tennis.csv, male_players.csv and female_players.csv. The dataset tennis.csv is full of matches with much information about tournaments, winners, losers and technical features related to how the players performed and scored. On the other hand, male_players.csv and female_player.csv are datasets with only two features: names and surnames of the players.[1]

With respect to the **first part** of this project, our goals are:

1. Create our own Database Schema from a reference figure given by the assignment using SQL Server Management Studio.

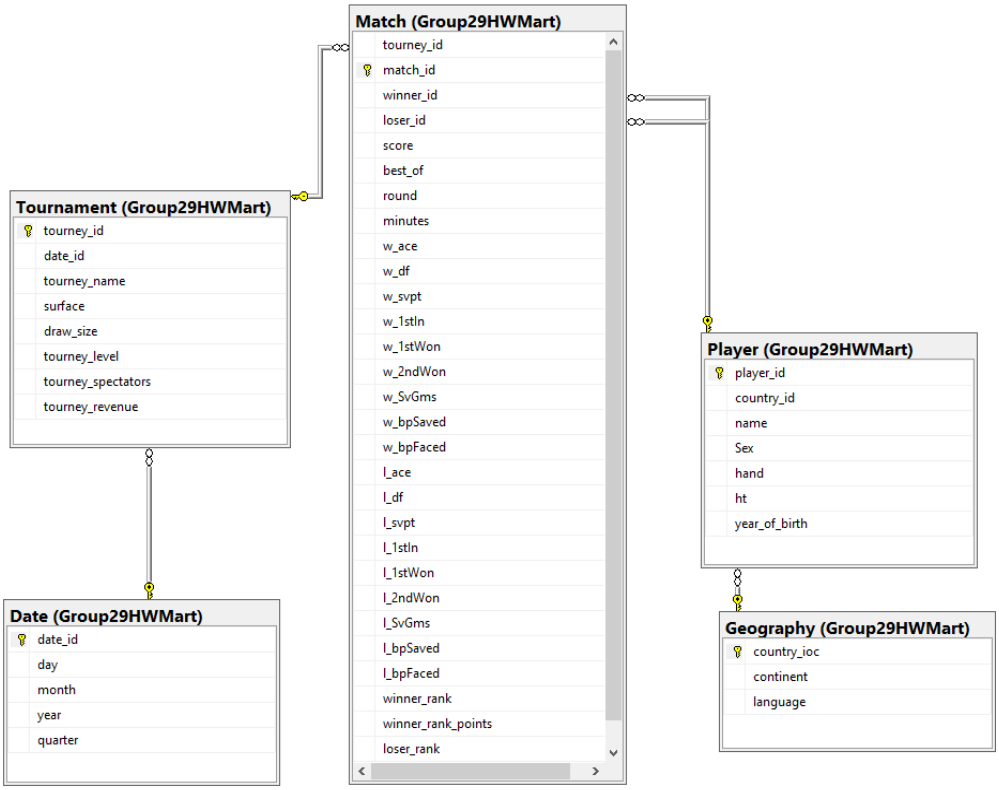
2. Write a python program that splits tennis.csv file into 4 chunks of different csv files with their proper names: "match", "player", "tournament", "date" and retrieving the feature "sex" from male_players.csv and female_players.csv.

3. Write a Python program that populates our database with the various tables from the .csv files, establishing schema relations as necessary.

[1] È possibile visualizzare tutte le eventuali modifiche rispetto alla prima consegna della prima parte di progetto, e seguenti, tramite il file di testo all'interno dello zip.

1.1 Assignment 0

Below is shown the database schema:



We also established relationships among the entities:

- From *Date* we have *date_id* as primary key that relates to *date_id* as foreign key in *Tournament*

- From *Tournament* we have *tourney_id* as primary key that relates to *tourney_id* as foreign key in *Match*

- From *Geography* we have *country_ioc* as primary key that relates to *country_id* as foreign key in Player

- From *Player* we have *player_id* that relates to both *winner_id* and *loser_id* as foreign keys in *Match*.

1.2 Assignment 1

The split of tennis.csv into match.csv, player.csv, tournament.csv and date.csv was done with csv library. We freely decided which columns where the most suited to be transferred into the different csv files and choose to execute the data cleaning part once we splitted everything.

The first pre-processing step was to create – with reference to the database schema – all the needed columns in each csv file. Pandas library was used to do so, almost everything was accomplished through the *.append()* method and *DataFrame.join()* method with **how='outer'**, which is a form union of calling frame's index (or column if on is specified) with other's index, and sort it, lexicographically.

```
# Creo le singole colonne finali per poi eliminare quelle da cui le ho derivate (non è certamente un metodo elegante, ma funzion

player_id = df_player['winner_id'].append(df_player['loser_id'], ignore_index=True).rename('player_id')
name = df_player['winner_name'].append(df_player['loser_name'], ignore_index=True).rename('name')
hand = df_player['winner_hand'].append(df_player['loser_hand'], ignore_index=True).rename('hand')
ht = df_player['winner_ht'].append(df_player['loser_ht'], ignore_index=True).rename('ht')
country_id = df_player['winner_ioc'].append(df_player['loser_ioc'], ignore_index=True).rename('country_id')
age = df_player['winner_age'].append(df_player['loser_age'], ignore_index=True).rename('age')
tourney_date =  df_player['tourney_date'].append(df_player['tourney_date'], ignore_index=True).rename('tourney_date1')

# Per evitare futuri problemi relativi alla struttura della tabella uso .join con outer
# Così avrò righe con tanti NaN ma a noi interessa che i valori delle colonne soprastanti abbiano il giusto 'match' tra di loro
# Cioè:  (Kei Nishikori --> country_id = JPN ---> è alto x ---> la sua mano è x ---> player_id=ecc...)

df1 = df_player.join([player_id,name,hand,ht,country_id,age,tourney_date], how='outer')
```

Figure 1: E.g., of player.csv's main columns creation

Some columns such as *match_id* were created through a simple concatenation. The *language* and *continent* features of geography's entity were obtained via the merging of countries.csv and languages.csv.

To create unique IDs for date_id in date table a LabelEncoder was imported from the sklearn.preprocessing library, assigning a numerical value to each different tourney_date (automatically discarding duplicates).

The sex feature was retrieved from male_players.csv and female_players.csv thanks to csv library, with the addition of a third sex value "Unknown" for all the unmatching/unfound players between the csv files, which was very important for not losing any player that was not present inside the above-mentioned csv files.

Day, month, year and quarter features of date.csv were created with the pandas method *.to_datetime*

Once all the needed columns were created, it was possible to deal with missing values and columns:

- All unneeded columns such as winner_name, etc… were deleted
- All the duplicates were managed with the drop_*.duplicates()* method that was used to drop all *player_id*, *date_id, tourney_id, match_id* , *country_ioc*) duplicates since they are the primary keys of our entities.
- The missing values of hand attribute were filled with 'U' which was one of the three unique values of the features

- The missing values of ht attribute were filled with the mean of all ht's values based on sex and approximated to the closest integer

```
# Fillo i NaN attraverso la media delle altezze per il sesso, approssimanto all'intero più vicino.

Player['ht'][(Player['Sex']=='Male')] = (Player['ht'].fillna(round(Player['ht'][(Player['Sex']=='Male')].mean())))
Player['ht'][(Player['Sex']=='Female')] = (Player['ht'].fillna(round(Player['ht'][(Player['Sex']=='Female')].mean())))
Player['ht'][(Player['Sex']=='Unknown')] = (Player['ht'].fillna(round(Player['ht'][(Player['Sex']=='Unknown')].mean())))
```

- The missing values of continent and language features were filled with 'Unknown'
- All the missing values in w_ace, w_svpt columns etc… were managed replacing them with "-1"
- The missing values of *age* attribute – which was used to get *year_of_birthday* feature – were filled with the mean of all age's values and rounded down, then the column was deleted.
- The missing values in winner_rank, loser_rank, winner_rank_points and loser_rank_points were filled with their own mean and rounded to the closest integer

With respect to the composition and order of columns of the database schema, all the attributes of all the csv files were reindexed properly.

1.3 Assignment 2

This second part was about populating the database tables with the various csv files, obtained from the previous assignment.

Firstly, a script has been used on SQL server management studio to create the 5 empty tables with the respective columns, then we used these tables to create the assignment: 0's diagram.

Once everything was settled a function was created and invoked to populate each table.

Everything was executed taking into account the foreign keys and primary keys constraints.

Chapter 2

In this chapter we explain how **SSIS** was used to solve some problems on the database previously created. The solutions had to be achieved without using any SQL command, but only through flows of nodes in ***Visual Studio***.

2.1 Assignment 0 – *For every country, the players ordered by number of matches won.*

To solve this problem we started by accessing the match table and picking only the needed column, such as *winner_id*, then with a **lookup node** we mapped the above-mentioned column with *player_id* and selected the *country_id* column. The last two steps are simply an **aggregate node** that execute a *Group by* of all the needed columns followed by a *Count* of all the same IDs that leads to the creation of "*matches_won*" column and a **Sort node** that re-orders the resultant columns as requested by assignment 0 before saving the file.

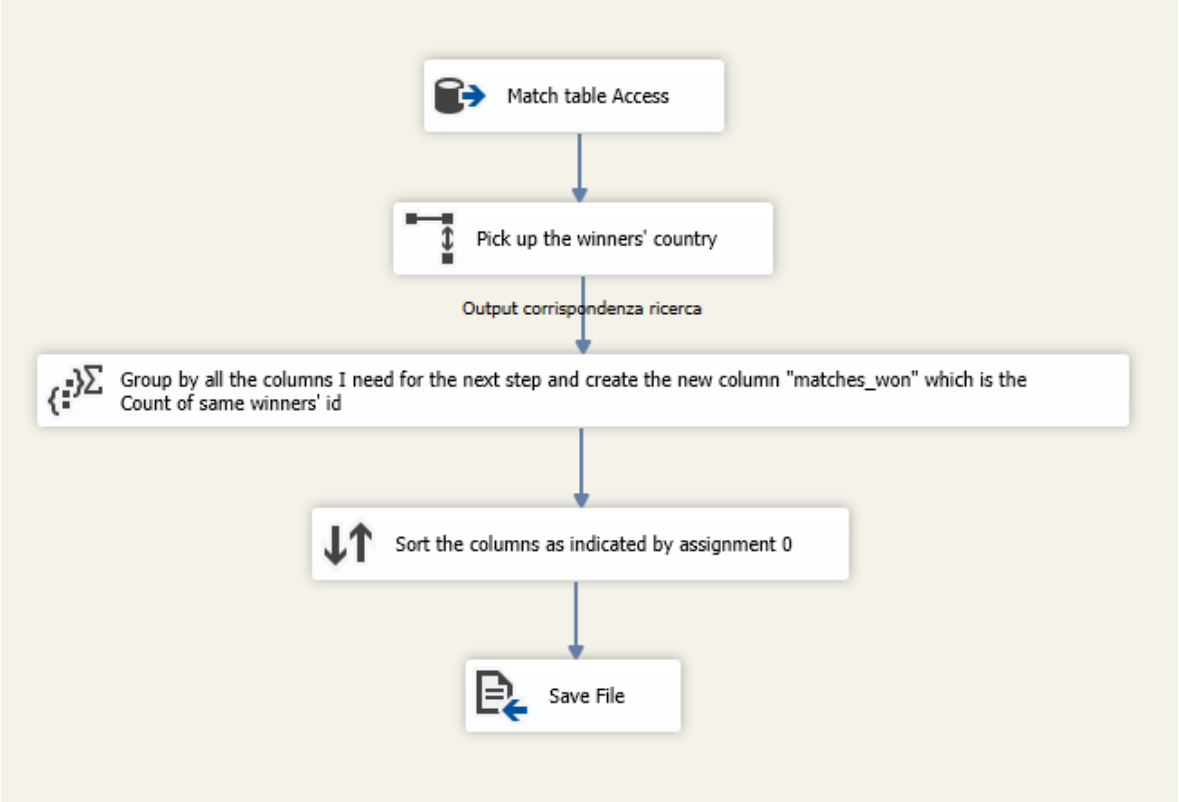


Figure1 - Assignment 0's data flow

2.2 Assignment 1 – *A match is said to be an "age mismatch" if the difference in years between the winner and the loser is more than 6.0. For each year, list the player that participated in the most age mismatches.*

For the beginning of this assignment, it was necessary to access the Match table and picking match_id, tourney_id, winner_id and loser_id, then we configurated **four lookup nodes**:

1. Looking up to winners' birth by mapping winner_id to player_id in Player table and selecting year_of_birth, then renaming it "winner_birth"

2. Looking up to losers' birth by mapping loser_id to player_id in Player table selecting and year_of_birth, then renaming it "loser_birth"

3. Looking up for date_id by mapping [Match]tourney_id to [Tournament]tourney_id in Tournament table

4. Looking up to year by mapping [Tournament]date_id to [Date]date_id in Date table.

After this first setup, a **conditional split node** was configurated with the following condition:

Ordine	Nome output	Condizione
1	Age_mismatch	ABS(winner_birth - loser_birth) > 6

Hence, it was possible to get only the age mismatched matches from this point onward.

Then, we needed to create a new column with all winners' and losers' ids for our next steps, we addressed this problem with the **Unpivot node** which takes the values of winner_id and loser_id and expand them into a new single column with more records, column that we named "PlayerID".

With all the players' id in one column it was finally possible to use **the aggregation node** and create the "age_mismatch" column from a Count of all match_id grouped by player_id and year.

The final part required the utilization of a **multicast node** to split the inputs into two different flows, so that we could operate different steps for the same data at the same time.

- On left side we create the new column "age_mismatch_max" from the max values of age_mismatch grouped by year

- On right side we just group by all the columns we need for the next steps.

Then we **sort the results and use a join node** with a single mapping from year[left] to year[right].

The last step was performed with a **conditional split node** with the following condition:

Ordine	Nome output	Condizione
1	top_mismatch	age_mismatch == age_mismatch_max

By doing so, it was possible to discriminate only those players with the highest number of age_mismatch for each year. In the end we save the file.

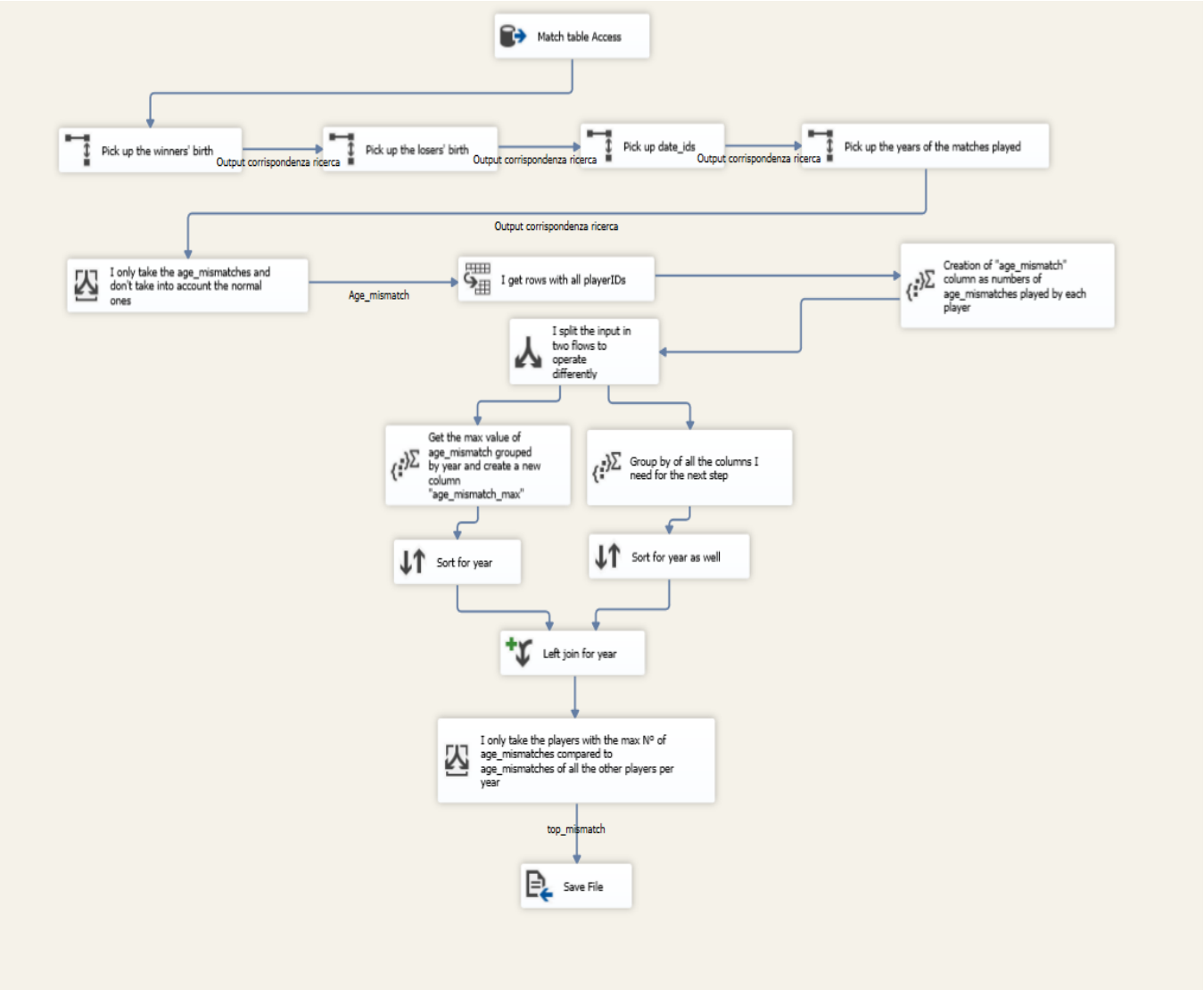


Figure 2 - Assignment 1's data flow

2.3 Assignment 2 – *Calculate for each player the total number of spectators that he performed in front of (assume for each match the same number of spectators as the respective tournament).*

This last assignment has seen the employment of the **Unpivot node** to get all winners' and losers' ids into a single column right after the Match table access. This was tremendously important since the aggregate node doesn't work properly with more than one column for a Group by, this first step resulted also into a minimal utilization of total nodes (five in total).

After getting all players' ids we use a **lookup node** to get the tourney_spectators column from Tournament table using tourney_id for the mapping (retrieved from Match table access).

In the end, before saving the file, we use the **aggregate node** where we execute a Count of tourney_spectators grouped by player_id to obtain the requested result.

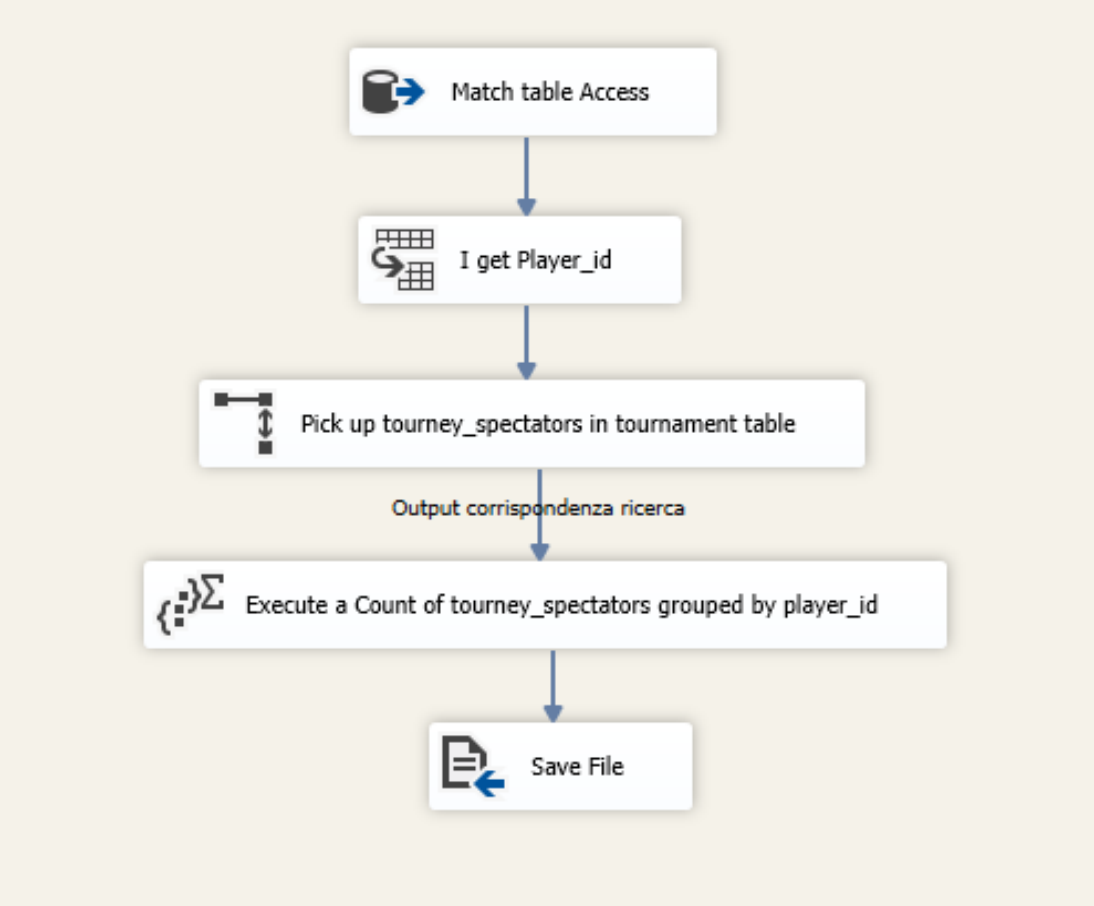


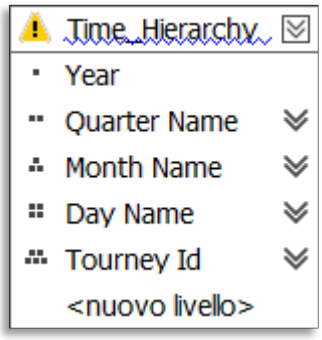
Figure 3 - Assignment2's data flow

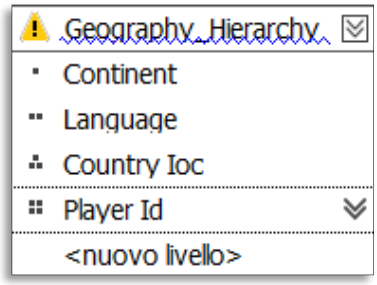
Chapter 3

In this chapter the generation of a datacube will be explained, then MDX queries will be used to interrogate it. At the end of the chapter there will also be some data visualization.

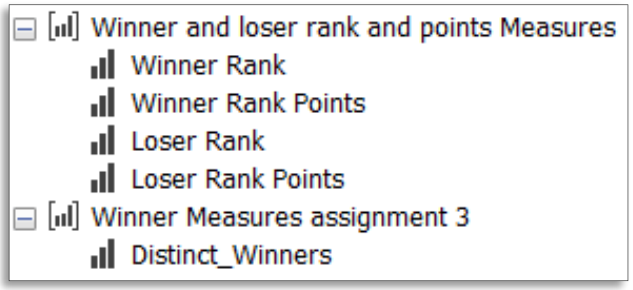
3.1 Assignment 0 - *Build a datacube from the data of the tables in your database, defining the appropriate hierarchies for time and geography. Use the rank and rank points of the winner and loser as measure.*

For the cube's creation the chosen dimensions were **Match**, **Player** and **Tournament** with *match_id*, *player_id* and *tourney_id* as primary keys. **Geography's** and **Date's** attributes were inserted inside Player's and Tournament's dimensions and were exploited for the construction of **Geography_Hierarchy** and **Time_Hierarchy**, as it is possible to observe in the following pictures:

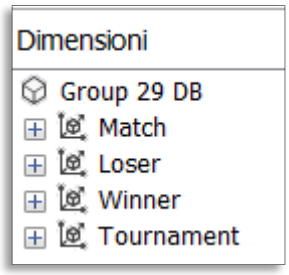




For **measures'** creation the attributes **winner_rank**, **winner_rank_points**, **loser_rank** and **loser_rank_points** from *match table* were exploited. In addition, it was necessary for an ulterior assignment (N°3) to create a new measure from the same table, which we called ***Distinct_Winners*** and it is a count of all the unique values of *winner_id* inside match table.



The resultant cube had the following dimensions:



It's very important to state that Winner and Loser dimensions were automatically inferred since the primary key is player_id from Player table, hence the **Player dimension** splitted itself into two different dimensions discerning winners and losers with their respective attributes (including also the geography hierarchies mentioned above)

After the cube is set, it was possible to interrogate it with MDX queries.

3.2 Assignment 1 - *Show the percentage increase in winner rank points with respect to the previous year for each winner*

This problem was addressed with the following query:

```
with member previous as
([Tournament].[Year].currentMember.lag(1), [Measures].[Loser Rank Points])

member perc as ([Measures].[Loser Rank Points] - previous)/ previous ,
format_string="percent"

select ([Tournament].[Year].[Year], perc) on columns,
NonEmptyCrossJoin([Match].[Loser Id].[Loser Id])
on rows from [Group 29 DB];
```

Messaggi | Risultati

	2016	2017	2018	2019	2020	2021
	perc	perc	perc	perc	perc	perc
100644	inf	145.09%	147.85%	21.66%	-61.26%	53.34%
101305	(Null)	(Null)	inf	-100.00%	(Null)	(Null)
101339	inf	-100.00%	(Null)	(Null)	(Null)	(Null)
102093	(Null)	(Null)	inf	0.00%	-95.51%	275.00%
102706	inf	-100.00%	(Null)	(Null)	(Null)	(Null)
102737	inf	-100.00%	(Null)	(Null)	(Null)	(Null)
102800	(Null)	inf	-100.00%	(Null)	inf	-100.00%
102863	inf	-100.00%	(Null)	(Null)	(Null)	(Null)
102864	inf	-15.24%	-100.00%	(Null)	(Null)	(Null)
102963	inf	-100.00%	(Null)	(Null)	(Null)	(Null)
102987	inf	0.00%	-100.00%	(Null)	(Null)	(Null)
103060	(Null)	(Null)	inf	-100.00%	(Null)	(Null)
103077	inf	-100.00%	(Null)	(Null)	(Null)	(Null)
103105	inf	8.07%	-100.00%	inf	-100.00%	(Null)
103163	(Null)	inf	-100.00%	(Null)	(Null)	(Null)
103175	inf	-98.00%	3172.73%	25.28%	-95.34%	-100.00%

http://lds.di.unipi.it/olap... | LAPTOP-F025QRR4\alfon | Group 29 DB | 00:00:05

3.3 Assignment 2 - *For each tournament show the total winner rank points in percentage with respect to the total winner rank points of the corresponding year of the tournament.*

This problem was addressed with the following query:

```
with member total_rank as
([Tournament].[Year].currentMember.lag(0), [Tournament].[Tourney Id].currentMember.parent,
[Measures].[Winner Rank Points])
member perc as [Measures].[Winner Rank Points]/total_rank, format_string="percent"

select {[Measures].[Winner Rank Points], total_rank, perc} on columns,
Nonemptycrossjoin([Tournament].[Year].[Year],[Tournament].[Tourney Id].[Tourney Id]) on rows
from [Group 29 DB];
```

Messaggi | Risultati

		Winner Rank Points	total_rank	perc
2016	2016-0083	12622	12875944	0.10%
2016	2016-0091	9028	12875944	0.07%
2016	2016-0213	11221	12875944	0.09%
2016	2016-0221	9424	12875944	0.07%
2016	2016-0228	12625	12875944	0.10%
2016	2016-0252	12789	12875944	0.10%
2016	2016-0300	56580	12875944	0.44%
2016	2016-0301	48621	12875944	0.38%
2016	2016-0308	32064	12875944	0.25%
2016	2016-0311	96706	12875944	0.75%
2016	2016-0314	22955	12875944	0.18%
2016	2016-0315	23237	12875944	0.18%
2016	2016-0316	25161	12875944	0.20%
2016	2016-0319	19724	12875944	0.15%
2016	2016-0321	45864	12875944	0.36%
2016	2016-0322	57328	12875944	0.45%
2016	2016-0328	72116	12875944	0.56%

http://lds.di.unipi.it/olap... | LAPTOP-F025QRR4\alfon | Group_29_DB | 00:00:03

3.4 Assignment 3 - *Show the winners having a total winner rank points greater than the average winner rank points in each continent by continent and year.*

The problem was addressed using the new measure "***Distinct_Winners***", with which we could discriminate the winners by continent and by year and then use them to calculate the average winner_rank_ points of each continent:

```
with member overall_points as
sum(([Tournament].[Year].currentmember,[Winner].[Continent].currentmember, [Winner].[Player
Id].currentmember.parent),
[Measures].[Winner Rank Points])

member Distinct_total_winners_ByContinent_ByYear as
```

```
(([Tournament].[Year].currentmember, [Winner].[Continent].currentmember, [Winner].[Player
Id].currentmember.parent),
[Measures].[Distinct_Winners])

member avg_continentPoints as overall_points/Distinct_total_winners_ByContinent_ByYear
,format_string = "#,##0.00"

select {Distinct_total_winners_ByContinent_ByYear,overall_points, [Measures].[Winner Rank
Points],avg_continentPoints} on columns,

filter(nonempty(([Tournament].[Year].[Year], [Winner].[Continent].[Continent],
[Winner].[Player Id].[Player Id])), [Measures].[Winner Rank Points] > avg_continentPoints)
on rows from [Group 29 DB];
```

Messaggi | Risultati

			Distinct_total_winners_ByContinent_ByYear	overall_points	Winner Rank Points	avg_continentPoints
2016	Africa	104291	54	98523	22711	1,824.50
2016	Africa	104731	54	98523	19585	1,824.50
2016	Africa	105434	54	98523	2481	1,824.50
2016	Africa	105633	54	98523	4756	1,824.50
2016	Africa	201618	54	98523	3142	1,824.50
2016	Africa	202460	54	98523	6525	1,824.50
2016	Africa	202581	54	98523	3434	1,824.50
2016	Africa	210640	54	98523	1880	1,824.50
2016	Africa	211529	54	98523	3038	1,824.50
2016	Africa	213972	54	98523	2414	1,824.50
2016	Africa	214565	54	98523	2350	1,824.50
2016	Africa	215041	54	98523	2841	1,824.50
2016	America	103607	572	2212181	15295	3,867.45
2016	America	104007	572	2212181	6988	3,867.45
2016	America	104122	572	2212181	24633	3,867.45
2016	America	104216	572	2212181	14684	3,867.45
2016	America	104268	572	2212181	6127	3,867.45

http://lds.di.unipi.it/olap... | LAPTOP-F025QRR4\alfon | Group_29_DB | 00:00:03

It was also possible to exploit the previously defined Geography_Hierarchy to solve this problem (differences with previous query are underlined):

```
with member overall_points as
sum(([Tournament].[Year].currentmember,[Winner].[Geography Hierarchy].currentmember.parent.p
arent.parent),
[Measures].[Winner Rank Points])

member Distinct_total_winners_ByContinent_ByYear as
(([Tournament].[Year].currentmember,
[Winner].[Geography Hierarchy].currentmember.parent.parent.parent),
[Measures].[Distinct_Winners])

member avg_continentPoints as overall_points/Distinct_total_winners_ByContinent_ByYear
,format_string = "#,##0.00"

select {Distinct_total_winners_ByContinent_ByYear,overall_points, [Measures].[Winner Rank
Points], avg_continentPoints} on columns,

filter(nonempty(([Tournament].[Year].[Year], [Winner].[Continent].[Continent],
[Winner].[Geography Hierarchy].[Player Id])), [Measures].[Winner Rank Points] >
avg_continentPoints)
on rows from [Group 29 DB];
```

However, as it is possible to appreciate in the following picture, the computational time is more than 5 times higher unlike the previous code (3 seconds vs 20 seconds):

Messaggi | Risultati

			Distinct_total_winners_ByContinent_ByYear	overall_points	Winner Rank Points	avg_continentPoints
2016	Africa	105633	54	98523	4756	1,824.50
2016	Africa	211529	54	98523	3038	1,824.50
2016	Africa	214565	54	98523	2350	1,824.50
2016	Africa	215041	54	98523	2841	1,824.50
2016	Africa	104291	54	98523	22711	1,824.50
2016	Africa	202460	54	98523	6525	1,824.50
2016	Africa	213972	54	98523	2414	1,824.50
2016	Africa	210640	54	98523	1880	1,824.50
2016	Africa	201618	54	98523	3142	1,824.50
2016	Africa	104731	54	98523	19585	1,824.50
2016	Africa	105434	54	98523	2481	1,824.50
2016	Africa	202581	54	98523	3434	1,824.50
2016	America	104330	572	2212181	8002	3,867.45
2016	America	104545	572	2212181	33995	3,867.45
2016	America	105023	572	2212181	12685	3,867.45
2016	America	105065	572	2212181	12439	3,867.45
2016	America	105385	572	2212181	19346	3,867.45

http://lds.di.unipi.it/olap... | LAPTOP-F025QRR4\alfon | Group_29_DB | 00:00:20

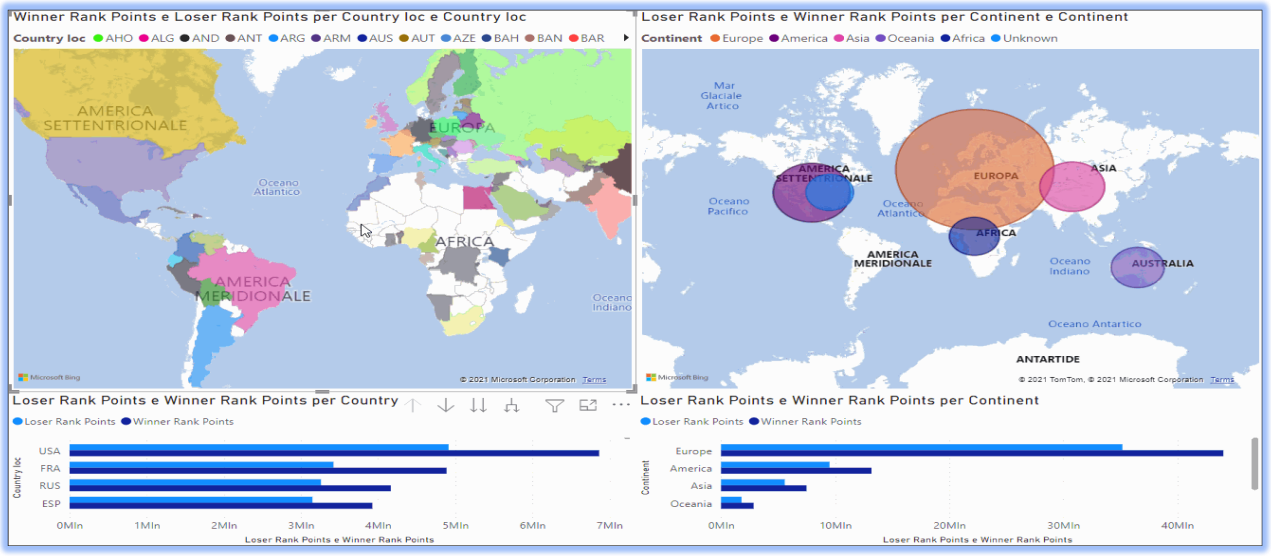
Of course, also the disposition of winner_id's results changed due to the hierarchy composition shown above in Assignment 0.

With that said, we decided to deliver only the query with the least computational time inside the final .zip file.

3.5 Assignment 4 - *Create a dashboard that shows the geographical distribution of winner rank points and loser rank points.*

Finally, it is requested to create a dashboard that shows the geographical distribution of the points, and to do so, we used PowerBI tool.

This dashboard includes two maps: **a coloured Map** with all the different countries, where it is possible to check their winners' and losers' rank points by passing with your own mouse cursor on the chosen country, **and a classic one** with the total points for each continent. The dashboard also includes two **grouped bar charts** linked to the two maps that shows the distribution of winner_rank_points and loser_rank_points in each country and continent.



3.6 Assignment 5 - *Create a plot/dashboard of your choosing, that you deem interesting w.r.t. the data available in your cube*

For this last assignment it was decided to create a dashboard showing the distribution of winners' and losers' rank point per continent and then per continent and surface. Also, a ribbon chart was used, displaying that even though USA solo made the greatest number of winners' and losers' rank points, America as a whole made lower than half total points in contrast to Europe each year (as seen in the line chart on the left side), showing that the sum of points of all the Europe's countries are by far greater than the ones in America.