Università di Pisa

Department of Computer Science
MSc in Data Science and
Business Informatics

Laboratory of Data Science Report

Tennis Data Warehouse

Authors

Gaetano Antonicchio 616685
Gloria Segurini 567352

2021/2022

Contents

1 Part 1 - Data-warehouse Construction

1.1 Assignment 0 - Database schema creation

The first assignment of the project asked us to design a star schema for the tennis data warehouse. After having analyzed the data types and attributes of the tennis.csv file, we created in SQL Management Studio the fact table for Match and all its dimensions, by specifying also the relationship between primary and foreign keys. As a principled design choice we imposed a NOT NULL constraint to all the primary keys present in the schema, while we allowed NULL values to the remaining ones.

In Figure 1, we show the resulting schema and its relations between fact and dimension tables:

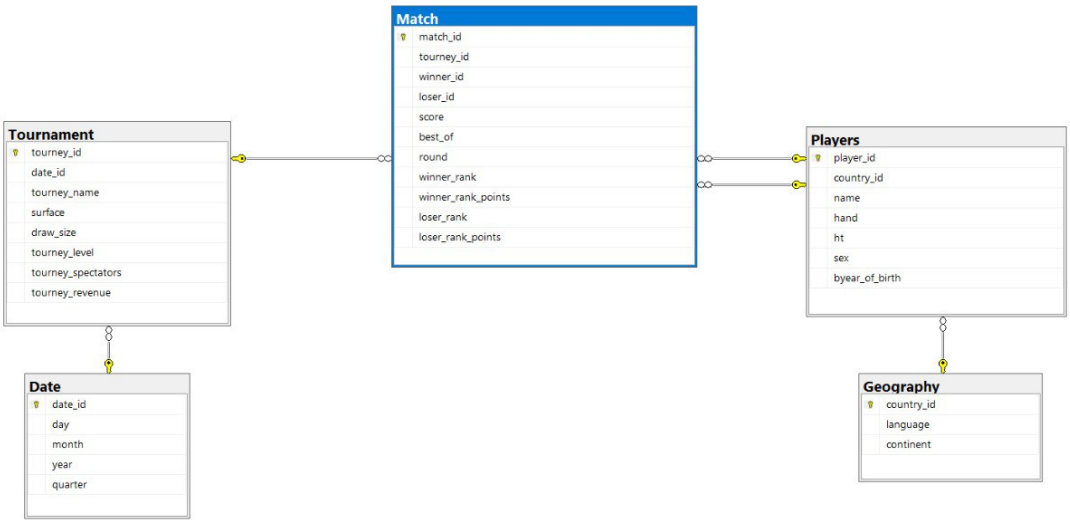

Fig. 1: Star schema - Tennis

1.2 Assignment 1 - Tables' creation

The fact and dimension tables, were produced by splitting the file tennis.csv into Match, Tournament, Players, Date and Geography. The coding part for this task was done in python using only the csv library when necessary. Pandas' library was only used in the post-processing step for handling missing values and fixing data inconsistencies. The four main scripts, namely: "splitTennis_match.py", "splitTennis_tournament.py", "splitTennis_date.py", "splitTennis_players.py" and "splitTennis_geography.py", were run from a "main.py" script which executed each one of them in one single command line.

In the following section, we describe the workflow followed for producing the requested tables.

1.2.1 Match

In order to produce the fact table "Match", we need to generate the primary key *match_id*. This corresponds to the concatenation of the variables *match_num* and *tourney_id*. It is important to mention that to assure the uniqueness for the key *tourney_id* in the tournament table, we decided to modify it, producing a new *tourney_id*, obtained by adding to it the variables *tourney_name* and *tourney_level*.
These changes were reflected also in the match table in such a way that the primary key in tournament matched the foreign key in match.
The fact table was obtained by iterating over a modified version of tennis.csv which included the *match_id* and the updated *tourney_id*. However the resulting table (which we named "match_beta.csv"), still presented duplicates and errors which needed to be addressed. The

duplicates were removed through the use of a set named "visited_id", which, at each iteration, keeps track of the newly encountered match_ids. During this phase, every row with a unique *match_id* is written to the output file match.csv.
Since in the raw match table (match_beta.csv) there are rows with the same match_id (hence apparently duplicate) but with different values for the remaining attributes, we decided to use a set "visited_rows" in which we store all the rows that are not being visited during the iterative reading process. This way, if at a given step one row has the match_id already present in the set "visited_id" but the remaining part of the row absent from the set "visited_rows" it means that the row refers to a different match and therefore needs to be added to the output file with a new (and unique) match_id. To select the rows that needs to be appended to the end of the output file, we added the entire row (whenever the conditions described above apply) to a set named "rows2write". Once we finished to read the match_beta.csv, we opened the output file match.csv in "a" mode, and we iterated over the rows2write set so to append each row with a new match_id obtained by concatenating to the existing one a progressive number. The resulting output file "match.csv" has 185764 rows and 30 columns.

1.2.2 Tournament

Tournament must consist of the following attributes: *tourney_id, date_id, tourney_name, surface, draw_size, tourney_level, tourney_spectators* and *tourney_revenue*. In order to create a discriminant primary key, *tourney_id* (as described in the Match section) is updated by concatenating to it the *tourney_name* and *tourney_level*. Since tournament is generated from tennis.csv, in which each row corresponded to a specific match played in a given tournament, the presence of duplicates in the output file "tournament.csv" (if not treated) was obviously certain. To avoid this, we used a set through which we performed a lookup operation on the *tourney_id*. The resulting file consists of 4911 rows and 8 attributes.

1.2.3 Date

The Date table contains the following attributes: *date_id, day, month, year* and *quarter*. In order to obtain them, it was sufficient to read tennis.csv and extract the required temporal information from the attribute *tourney_date*, which was provided in the form DDMMYYYY. The variable quarter which has only 4 unique values (Q1, Q2, Q3, Q4), was obtained through an if statement that checked the number of the month (i.e. if a row has a month in range from 1 to 3 the corresponding quarter is Q1). In total date.csv is composed of 375 rows and 5 columns.

1.2.4 Players

In addition to the attributes already included in tennis, the dimensional schema players requires the "sex" attribute. This information is obtained through a data integration with the datasets: male_players.csv and female_players.csv. However, before proceeding with the integration phase, we cleaned the csv files from errors, NULL or Unknown names and non-existing names (i.e. X, X). In order to get the sex, we constructed a dictionary called "players" in which we stored as a key the name+surname of each players (in a way to make it possible to do a cross-comparison with tennis.csv, in which the name and surname of each players are saved in a single variable) and as values the sex, which takes value "M" if the player name has been extracted from male_players.csv and value "F" if it was obtained from female_players.csv. Since some players were present in both csv, hence they resulted having both sexes, we decided to keep 2 sets (one for male and one for female) in which we stored the names of the players processed from the 2 files. After we constructed the dictionary, we computed the intersection between the 2 sets in order to identify those players with uncertain sex. This attribute was correctly updated for 6 out of 78 by looking at the sex of its opponents. If the name of the opponent was not included in the intersection (which means that its sex was assigned correctly) then the player for which we wanted to determine the sex must've had the same sex (because in tennis, matches mixed gender games are not admissible). Regarding the attribute "hand", we noticed that some players had the

value "U" (Undefined), even though in some of their matches they used other hands. To get the correct hand, we built a dictionary in which we stored all the hands used by those players having hand U. We then selected the most used hand different than U, and for those that had the value U in every match, we considered it as NULL. The *byear_of_birth* instead, has been computed taking the difference between the year (extracted from *tourney_date*) and the age at which the player played the game. In case the age was NULL, since we couldn't take the subtraction, we considered it as NULL. To extract the players' attributes from tennis.csv, we used 2 lists of lists as data structures, one for winners and one for losers, in which we only saved the attributes of interests. Afterwards we took advantage of 2 sets visited_w (for winners) and visited_l (for losers) used to seed up the running time of the code, other 2 sets w_names and l_names in which we stored the names of winners and losers (used to determine the name of those players present in tennis but absent from male_players.csv and female_players.csv) and a dictionary players_final in which we store all the needed information that need to be written to the output file. By using the sets, we identified 30 players with sex NULL that needed to be added to the output file. We then performed a manual check on the source https://www.itftennis.com for determining the sex of those players. We discovered that out of 30 only 1 player, namely "Alona Fomina" had sex F. After the dictionary was populated, we transferred its information on the file player.csv by opening the file in write mode. In total, players.csv has 10105 rows and 7 attributes.

1.2.5 Geography

In order to generate the geography table, we started with the csv file "countries" and then we performed data integration using country_list.csv, obtained from the source:
http://www.fullstacks.io/2016/07/countries-and-their-spoken-languages.html.
This additional dataset provided us information about the language spoken in each of the listed countries. Before integrating the data, we noticed that some pre-processing was necessary on country_list, due to the fact that the name of countries (which were used as a key for matching the two csv) presented typing error (i.e. Rí©union instead of Reunion), or had unmatching names (i.e. United Kingdom in country_list.csv and Great Britain in countries.csv) or had wrong languages assigned to some countries, for example, according to country_list.csv the spoken language in Canada was the Hawk, but in reality it is actually English,so we manually changed to English. Afterwards, we opened the csv file countries and started reading it row by row. A check on the *country_ioc* (used as key) and a set were used to remove duplicate lines. Furthermore, some country name in countries.csv were misspelled (i.e. Urugay instead of Uruguay); therefore, when in the for loop the key pointed to one of those countries with syntactic errors, we corrected the country name so that it could match with the one in country_list. We also removed duplicate countries with different *country_ioc*: for example we noticed that *country_ioc* ITF and ITA both pointed to the same nation: Italy. One row in country_list.csv instead, had only the *country_ioc* (POC) and a series of NULL values. These values were integrated with information provided on wikipedia.it. In fact, we discovered that POC referred to Pacific Oceania which, in tennis tournaments, groups all the Oceania countries into one nation. After gaining this information, we added the related language and continent. After we wrote the geography file, we noticed that in players.csv we had players with a *country_ioc* that was not present neither in countries.csv nor in country_list.csv. In order to identify those missing countries, we used a set subtraction of *country_id* sets (one calculated over geography.csv and the other one over players.csv). In total we discovered 29 missing *country_id*, that we appended to the file geography.csv with the related language and continent information, which were obtained after a manual search on wikipedia.it. The resulting geography.csv file is composed of 153 rows and 3 columns.

1.3 Post-processing: missing values duplicates

1.3.1 Match table

We observed that the Match table presented attributes with more than 80% of missing values. These were statistics related to the game which couldn't be traced or restored by looking at the other

variables in the data. Since removing the rows having missing values would have drastically reduced the number of available data, we decided to remove these attributes instead. Below we list the attributes that were removed from the fact table Match: *'minutes', 'w_ace', 'w_df ', 'w_svpt', 'w-1stIn', 'w 1stWon', 'w_2ndWon', 'w_SvGms', 'w_bpSaved', 'w_bpFaced', 'l_ace', 'l_df ', 'l_svpt', 'l_1stIn', 'l_1stWon', 'l_2ndWon', 'l_SvGms', 'l_bpSaved','l_bpFaced'*.
After removing these columns, we still needed to deal with attributes having less than the 50% of NULL values. We were able to retrieve information about *winner_rank* and *loser_rank* by performing a group by *year* and *player_id* whose aggregate function mean(*winner_rank* or *loser_rank*) was used to replace the NULL values. However this operation didn't fill all the missing values in the dataset. We then decided to keep the rows with NULL values for not losing any other information about the matches. The attributes that were loaded with missing values are: *score*: 169, *winner_rank*: 15429, *winner_rank_points*: 19388, *loser_rank*: 29423, *loser_rank_points*: 35248. Lastly, we found and removed 6 matches (rows) in which each player played against himself/herself. In total, the match.csv after the post-processing step counted 185758 rows and 11 columns.

1.3.2 Tournament table

The table Tournament presented 62 missing values for the attribute *surface*. This column admits only the following values: 'Hard', 'Clay', 'Grass', 'Carpet'. After taking a look to the distribution of this variable, we decided to use the mode, which applied the value "Hard" to all the NULL values in surface.

1.3.3 Players table

The table Players presents 6276 missing values for the attribute *hand*, 9550 for *ht* and 2109 for *byear_of_birth*. The NULL values in *hand* have been replaced using the mode, which returned the value "R". In fact 89.36% of players use the right hand when playing (3394 players) while the remaining 10.64% uses the left hand (404 players). The *ht* (height) has been replaced using the mean calculated over the *sex* and the *continent*. We decided to use the *continent* as aggregate for the group by instead of *country_id* because, even if it would have been more accurate, the latter had some countries with only players with NULL *ht* (hence the replacement wouldn't have had effect). We also checked the data distribution before and after the changes, and we noticed that the overall distribution didn't present noticeable changes, therefore we assumed that the technique we used was acceptable. We also noticed that 2 players (Kamilla Rakhimova and Ilija Vucic) had unusual *ht* values. In fact the player Kamilla Rakhimova resulted to be tall 2.0 cm while Ilija Vucic was 145.00 cm tall. These players were identified by filtering all players tall less than 150 cm. The height of Ilija Vucic was rapidly replaced (188.0 cm was the correct value) using the information provided by the source https://www.atptour.com/en/players/ilijavucic/v624/overview, while for the other one we weren't able to find information on the external source, therefore we simply changed the NULL value to the average height of European females (173.00 cm).
The field *byear_of_birth* instead couldn't be determined using the available data neither with external sources. Instead of removing those players from the table, we decided to keep them. However since the loading of the table on SQL Management Studio generated errors on the data type, we decided to assign the value -1 to those players that had NULL values in *byear_of_birth*, and then we executed the following SQL query to restore the values from -1 to NULL.

```
--- Change -1 values to NULL  ---
update Players set byear_of_birth = NULL where byear_of_birth = -1
```

Fig. 2: SQL query used for changing values from -1 to NULL.

1.4 Assignment 2 - Loading data into the DB

The loading of the data on SQL Management Studio was performed by running separate python scripts one for each table. In each file .py we established a connection to the remote Database and then we run the query INSERT INTO Table VALUES using a cursor which was then closed at the end of the loading process. Since we had connection problems related to the VPN connection, we decided to split the largest file (like match.csv) into smaller chunks of 3000 rows each. This way the commit was performed after each successful loading. This strategy avoided us to reload the whole dataset if problem occurred during this step.

2 Part 2 - SSIS Solutions

2.1 SSIS Data Flows

In order to construct the data flows required to answer the business questions we were assigned, we first opened Visual Studio 2019 and created an Integration Service Project named Task2_Group28. Afterwards, we instantiate three packages: assignment0_group28.dtsx, assignment1_group28.dtsx and assignment2_group28.dtsx.

2.1.1 Assignment 0

The assignment asks us to return for every country, the players ordered by number of matches won.

The proposed solution requires first the reading of the Match table from the Database "Group 28 DB" through a "Origin OLE DB" node.
From this table we selected only one column: *winner_id*, which plays the role of foreign key to the table Players. We then did a lookup on the table Players using the join condition: *winner_id* = *player_id*. This allowed us to extract from Players the columns: *name* and *country_id*.
We then used the group by node to count the number of matches won by each players. To do this, we simply needed to group the attributes on *country_id*, *winner_id* and *name* and compute as aggregate function, the count(*) (renamed *NumMatchWon*) which provided us for each player the number of matches won. After this, we only needed to order the results by *country_id* (in increasing order) and *NumMatchWon*(in decreasing order) using the order by node.
The final result was then saved to a .txt file using the Destination File Flat, through which we specified the destination folder and the name of the file, other than the format of the output.

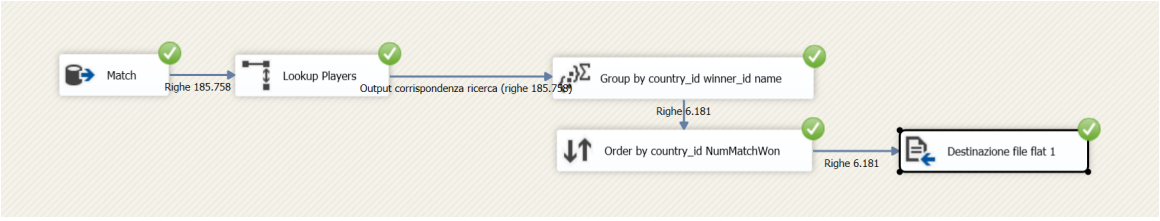

Fig. 3: Data Flow assignment 0

2.1.2 Assignment 1

The assignment asks us to return for each year, the list of players that participated in the most age mismatches.
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.

Firstly, we get the following attributes from Match table: *tourney_id*, *winner_id* and *loser_id*. Then, we did a first lookup on Players, imposing a mapping condition between *winner_id* and *player_id*; we

also got the attributes *byear_of_birth* and *name*, which were renamed respectively *byear_of_birth_winner* and *winner_name* and added as new columns. We did the same on the next lookup on Players, this time for losers, getting the matching condition between *loser_id* and *player_id* and the new columns *byear_of_birth_loser* and *loser_name*.

Secondly, we imposed the splitting condition, which allowed us to get all the NOT NULL *byear_of_birth* both for winners and losers and also imposing the age_mismatch condition which states as follows: **!ISNULL(byear_of_birth_winner) && !ISNULL(byear_of_birth) && ABS(byear_of_birth_winner - byear_of_birth_loser) > 6**. When dragging the blue arrow from the conditional split to the next node, it is strictly needed to select *age_mismatch* as output: as explained in the SSIS documentation, the conditional splitting node allows to insert the data satisfying the stated conditions into the table [age_mismatch]. Moving on, we made a lookup on Tournament, taking the *date_id* attribute as new column, which was then used for the mapping in the next lookup on Date, from which we took *year* as new attribute.

In order compute the number of mismatches, we needed to consider the times in which an age mismatch occurred for the player both when he was winner and when he was loser: we first did a multicast, so as to group by *year, winner_id, winner_name* and computing a COUNT ALL on one side and did the same for losers on the other one. In this step, we also renamed both *winner_id* and *loser_id* as *player*. At this point, we used the node for multiple inputs union, which works as a UNION ALL operator on SQL, getting single columns for *year*, *player*, *player_name* and the COUNT ALL column. We could have decided not to add the information about the player's name, but since this is an informative report, we decided to add a significant attribute to better recognize the player.

We moved on doing another group by on *year, player, player_name* and summing the COUNT ALL which we renamed as *tot_num_mismatches*. We then applied another multicast node as we needed to perform a group by with *year* as aggregation attributes, and MAX(*tot_num_mismatches*) as aggregation function. This allowed us to determine the max number of mismatch per year. After this operation, we ordered the output coming from the multicast by *year* and *tot_num_mismatches* and the second one (coming from the group by) by *year* and *MAX_tot_num_mismatches* so as to do a MERGE JOIN on *year* and on *tot_num_mismatches* and *MAX_tot_num_mismatches*. In this way, the output provided the *year*, *player_id*, *player_name* and *MAX_tot_num_mismatches*. The result was ultimately saved on a .txt file using the Destination File Flat node.

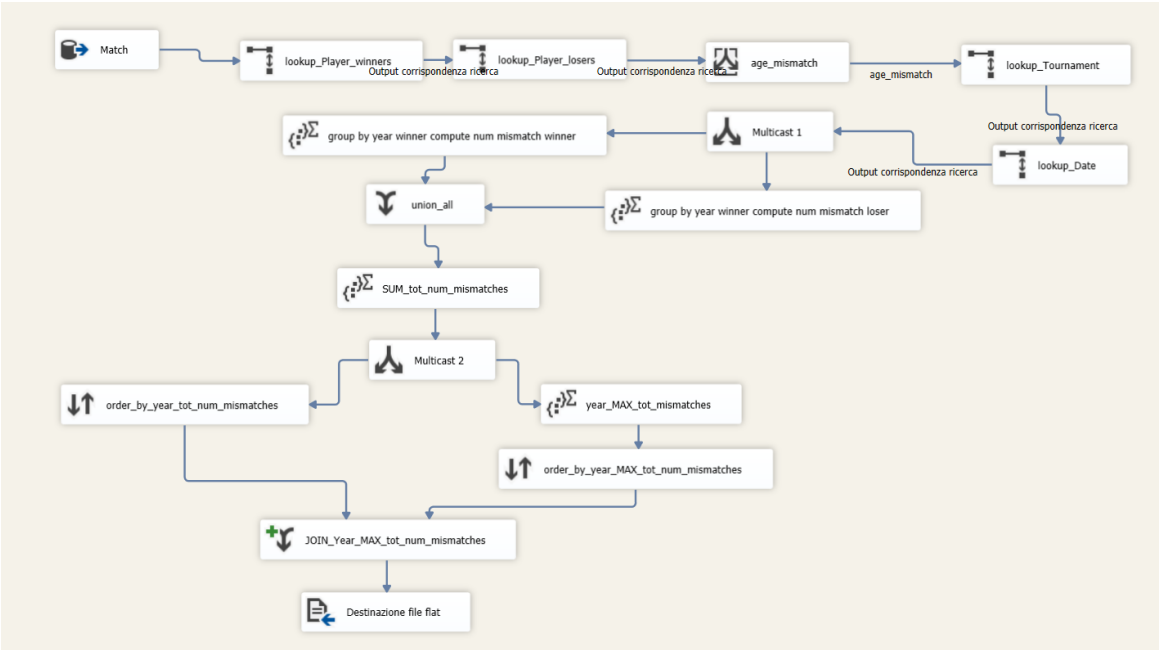

Fig. 4: Data Flow of Assignment 1

2.1.3 Assignment 2

The assignment asks us to calculate for each player the total number of spectators that he performed in front of.

We first used the Origin OLE DB node to read Match with the following selected columns: *winner_id*, *loser_id* and *tourney_id*. We then used a lookup on Tournament to get the additional column *tourney_spectators*, needed for answering the question.

We then performed two lookup on the Player table, one for winners and one for losers. Through this operation we could extract the columns *winner_name* and *loser_name*. This lookup was needed only for a better readability of the result. If instead in the output file we wanted to display only the *player_id*, we could have avoided these two lookups.
A multicast node was necessary as we wanted to apply different transformations on the same output in parallel. On one side we performed a group by on *winner_id* and *winner_name* summing up the total number of spectators for winners, and on the other side we did the same for losers. With the union all operator we combined the data coming from the two different sources described above (winners and losers). The input coming from the columns *winner_id* and *loser_id* were renamed as *player_id*. This allowed us to perform a group by on *player_id* in order to sum up the total number of spectator a player performed in front of both when he/she was a winner and loser. After this step, we saved the result to a .txt file through a Destination Flat File node.

It was also possible to perform this task reading the Match table twice,taking first *winner_id* and *tourney_id* and then *loser_id* and *tourney_id*, so as to perform two distinct lookup on Tournament. However, this would have been more expensive. We decided to show this alternative too in solution in Figure 6.

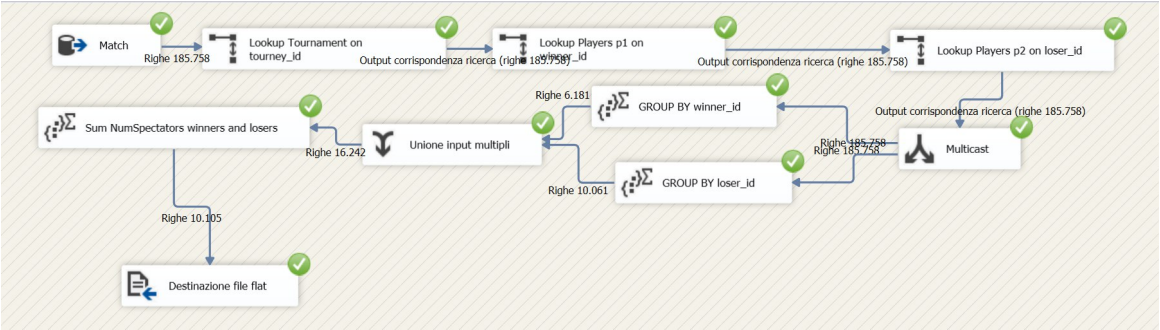

Fig. 5: Data Flow assignment 2

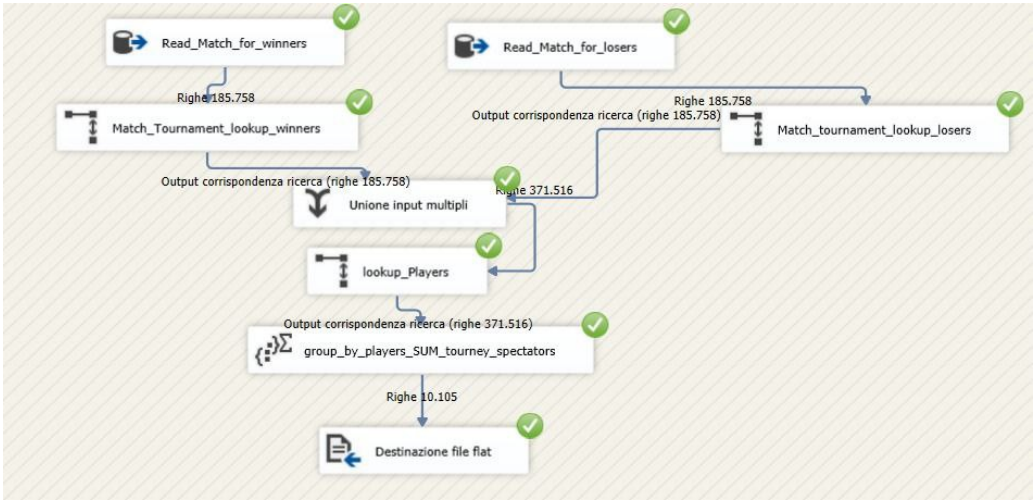

Fig. 6: Data Flow assignment 2 - Alternative solution

3 Part 3 - Multidimensional Data Analysis

3.1 OLAP Cube

After having established a connection with the server side, we selected the fact table *"Match"* and the related dimensions. This allowed us to have a view on the database Group_28_DB. Before creating the OLAP cube, we created the dimensions Tournament and Players. In the first one, we included also the attributes related to the Date table, as the former point to the latter with a foreign key relation on *date_id*. Within this dimension we included the hierarchy renamed "Time_Hierarchy" having the following relations: Year → Quarter → Month → Day → Tourney Id. In order to guarantee the order between years, months and days, we set in Property - Advanced, the variable "OrderBy" to the value "key" (because the values were all integers). In the Players dimension instead we created the "Geography_hierarchy" with the following structure: Continent → Country Code → Player Id. The mentioned hierarchies are shown in Figure 7.
After this step we generated the OLAP cube, including in the fact table "Match" the measures needed for the analysis: *winner_rank*, *winner_rank_points*, *loser_rank* and *loser_rank_points*. For those we set the ArggregateFunction to "Sum" and the FormatString to "Standard". Furthermore, we pre-computed the measure "NDistinct_winners" (obtained using the AggregateFunction distinct count on the attribute *winner_id*) which was used to solve the query 3 in MDX. Since we wanted additional measures for performing a more in-depth analysis through dashboards, we also added measures such as: "TOT_spectators" (obtained by applying as AggregateFunction the sum on the attribute *tourney_spectators* and selecting as FormatString the value "Standard"), "TOT_revenue" (obtained in a similar way, this time on the attribute *tourney_revenue* with a FormatString set to "Currency"),lastly "NDistinct_losers" (which was obtained in a similar way to NDistinct_winners). Once the cube was ready, we performed the deployment.

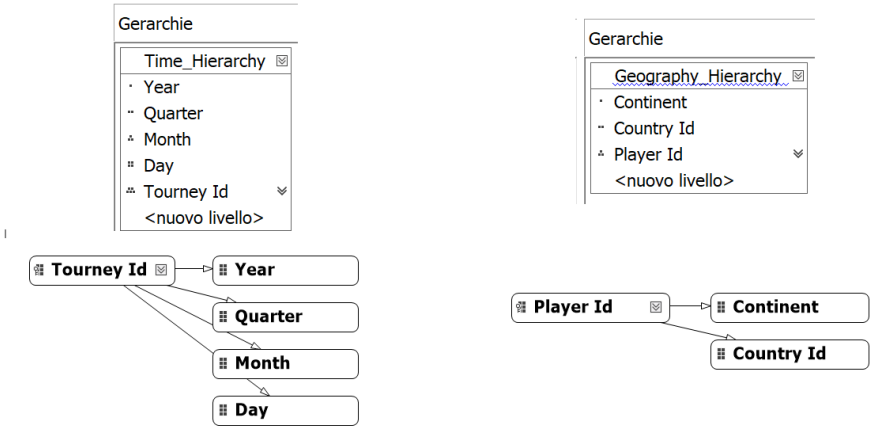

Fig. 7: Time_Hierarchy

3.2 Query MDX

The following MDX queries are saved in the file named "MDXQueries_task3_group28.mdx".

3.2.1 Query 1

Show the percentage increase in loser rank points with respect to the previous year for each loser.

Firstly, we put the tuple containing the Player Id and the Year in the rows. Secondly, we calculated the previous year's loser rank points using the PrevMember operator (alternatively we could have used ".lag(1)", which would have yielded the same result). In this way we were able to calculate the percentage variation considering the difference between the Loser Rank Points and the previous year's ones, divided by the latter. Moreover, we used a **iif** statement to show the label 'N/A' for

those records presenting a *amount_prevYear* equal to null so as to avoid "inf" values in the output. Furthermore, since we wanted to display a percentage, we set the format string to the value "percent". In conclusion, we put a nonempty on the rows so that only Loser Rank Points different from null were shown in the result.

```
with member amount_prevYear as(
   [Tournament].[Year].currentmember.PrevMember,
   [Measures].[Loser Rank Points])

member percentage_variation as
iif (amount_prevYear = null, 'N/A',
   ([Measures].[Loser Rank Points] - amount_prevYear) / amount_prevYear),
format_string='percent'

select {[Measures].[Loser Rank Points], percentage_variation} on columns,
nonempty(([Loser].[Player Id].[Player Id],[Tournament].[Year].[Year])) on rows
from [Group 28 DB]
```

Fig. 8: Query 1 - MDX

Messages Results

		Loser Rank Points	percentage_variation
100644	2016	26,693.00	N/A
100644	2017	68,830.00	157.86%
100644	2018	95,980.00	39.45%
100644	2019	116,770.00	21.66%
100644	2020	45,240.00	-61.26%
100644	2021	69,370.00	53.34%
101305	2018	0.00	N/A
101339	2016	0.00	N/A
102093	2018	0.00	N/A
102093	2019	0.00	N/A
102093	2020	16.00	N/A
102093	2021	60.00	275.00%
102706	2016	0.00	N/A
102737	2016	0.00	N/A
102800	2017	0.00	N/A
102800	2020	0.00	N/A
102863	2016	24.00	N/A

Fig. 9: Query 1 - Result Preview

3.2.2 Query 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.

The first step was to create a member "Tot_winnerRankPoints_byYear" which returned the total winner rank points for each year. Since in the select the Time hierarchy pointed to *tourney_id* we used the parent method to navigate the hierarchy upwards until we reached the granularity "Year". We then computed another member named "percent_composition" which was calculated dividing the measure *Winner Rank Points* by the "Tot_winnerRankPoints_byYear". In **format_string** we specified the string "percent" as we wanted a percentage in output. In the select we projected Tot_winnerRankPoints_byYear, the measure *Winner Rank Points* which was renamed "WinnerRankPoints_byTournament" and the "percent_composition". Since we wanted to see the percentage contribution of each tournament to Winner Rank Points in that given year, we selected The flat hierarchy Year, and Tourney_id, which was extracted from the Time_hierarchy.

```
with member Tot_winnerRankPoints_byYear as
([Tournament].[Time_Hierarchy].currentmember.parent.parent.parent.parent,
[Measures].[Winner Rank Points])

member percent_composition as
[Measures].[Winner Rank Points]/Tot_winnerRankPoints_byYear,
format_string='percent'

select {Tot_winnerRankPoints_byYear, [Measures].[Winner Rank Points] as WinnerRankPoints_byTournament ,
        percent_composition }on columns,
([Tournament].[Year].[Year],[Tournament].[Time_Hierarchy].[Tourney Id]) on rows
from [Group 28 DB]
```

Fig. 10: Query 2 - MDX

Messages Results

		Tot_winnerRankPoints_byYear	Winner Rank Points	percent_composition
2016	2016-0083eckental chC	15616213	11,212.00	0.07%
2016	2016-0091kyoto chC	15616213	9,028.00	0.06%
2016	2016-0213san luis potosi chC	15616213	10,751.00	0.07%
2016	2016-0221tampere chC	15616213	6,604.00	0.04%
2016	2016-0228winnetka chC	15616213	11,685.00	0.07%
2016	2016-0252medellin chC	15616213	11,849.00	0.08%
2016	2016-0300luxembourgI	15616213	56,580.00	0.36%
2016	2016-0301aucklandA	15616213	48,621.00	0.31%
2016	2016-0308munichA	15616213	32,064.00	0.21%
2016	2016-0311queen's clubA	15616213	96,706.00	0.62%
2016	2016-0314gstaadA	15616213	22,955.00	0.15%
2016	2016-0315newportA	15616213	23,237.00	0.15%
2016	2016-0316bastadA	15616213	25,161.00	0.16%
2016	2016-0319kitzbuhelA	15616213	19,724.00	0.13%
2016	2016-0321stuttgartA	15616213	44,924.00	0.29%
2016	2016-0322genevaA	15616213	57,328.00	0.37%
2016	2016-0328baselA	15616213	72,116.00	0.46%

Fig. 11: Query 2 - Result Preview

3.2.3 Query 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 first step was to compute the Total number of winner rank points by continent and year. This was possible with a member "sum_point_byYear_byContinent" which was obtained using the **sum** function. The total was generated by fixing the "Time_Hierarchy" and "Geography_Hierarchy" and summing the related *winner_rank_points*. To indicate that we wanted to sum the points by Year and Continent, we used the **currentmember** and **parent** method to travel the hierarchy until we reached the desired slice. We then computed another member "total_winnersByContinentYear" which used the pre-computed measure "NDistinct_winners" that was created along with the cube. By fixing the temporal and spatial dimension, we counted the total distinct winners in each year and continent. This represented the denominator of the division computed in the member "avg_points" which returned the average *winner rank points* in each continent for each year. In the select instead we projected on the columns the variables: "sum_point_byYear_byContinent", "avg_points" and *winner_rank_points*. On the rows, we applied a **filter** function in order to select those players whose had the total winner_rank_points greater than the average *winner_rank_points* of that *continent* and *year*. To avoid displaying "(null)" values, we used the **nonempty** function within the filter.

```
with member sum_points_byYear_byContinent as
sum(([Tournament].[Time_Hierarchy].currentmember,
[Winner].[Geography_Hierarchy].currentmember.parent.parent),
[Measures].[Winner Rank Points])

member total_winnersByContinentYear as
(([Tournament].[Time_Hierarchy].currentmember,
[Winner].[Geography_Hierarchy].currentmember.parent.parent), [Measures].[NDistinct_winners])

member avg_points as
sum_points_byYear_byContinent/  total_winnersByContinentYear

select {sum_points_byYear_byContinent, avg_points, [Measures].[Winner Rank Points] , total_winnersByContinentYear} on columns,
filter(nonempty(([Tournament].[Time_Hierarchy].[Year], [Winner].[Continent].[Continent],
            [Winner].[Geography_Hierarchy].[Player Id])),[Measures].[Winner Rank Points]> avg_points ) on rows
from [Group 28 DB]
```

Fig. 12: Query 3 - MDX

Messages Results

			sum_points_byYear_byContinent	avg_points	Winner Rank Points
2016	Africa	105633	80,821.00	1,524.92	4,756.00
2016	Africa	205909	80,821.00	1,524.92	1,645.00
2016	Africa	211529	80,821.00	1,524.92	3,038.00
2016	Africa	201618	80,821.00	1,524.92	3,142.00
2016	Africa	104731	80,821.00	1,524.92	26,985.00
2016	Africa	202581	80,821.00	1,524.92	3,434.00
2016	Africa	104291	80,821.00	1,524.92	26,754.00
2016	Africa	202460	80,821.00	1,524.92	6,525.00
2016	America	104122	2,502,622.00	4,367.58	25,095.00
2016	America	104216	2,502,622.00	4,367.58	14,684.00
2016	America	104338	2,502,622.00	4,367.58	9,205.00
2016	America	104547	2,502,622.00	4,367.58	29,192.00
2016	America	104919	2,502,622.00	4,367.58	19,880.00

Fig. 13: Query 3 - Result Preview

3.3 Dashboard

3.3.1 Power BI Solution

3.3.2 Assignment 1

We first established a connection to the server clicking on the Analysis Services option and inserting the connection string *http://lds.di.unipi.it/olap/msmdpump.dll*. After selecting out Group_28_DB Cube, Power BI showed us the Cube's structure on the right side. This allowed us to draw the line plot shown in Figure 14.

We plotted the geographical distribution of the Loser and Winner Rank Points. To do this we used both a map and a line graph. In the Dimension we put the measures Winner Rank Points and Loser Rank Points, whereas in the Location we selected the hierarchy "Geography_hierarchy" (selecting as granularity, only the continent and country_id). The same hierarchical attributes were used in the Legend to better distinguish the observations by continent and country. From the map we noticed that the highest Winner and Loser Rank Points were recorded in Europe, followed by America, Asia, Oceania and lastly Africa. Observing the granularity by country instead, we noticed that The US had the highest Winner Rank Points (7.051.285,00) and Loser Rank Points (5.368.528,00).

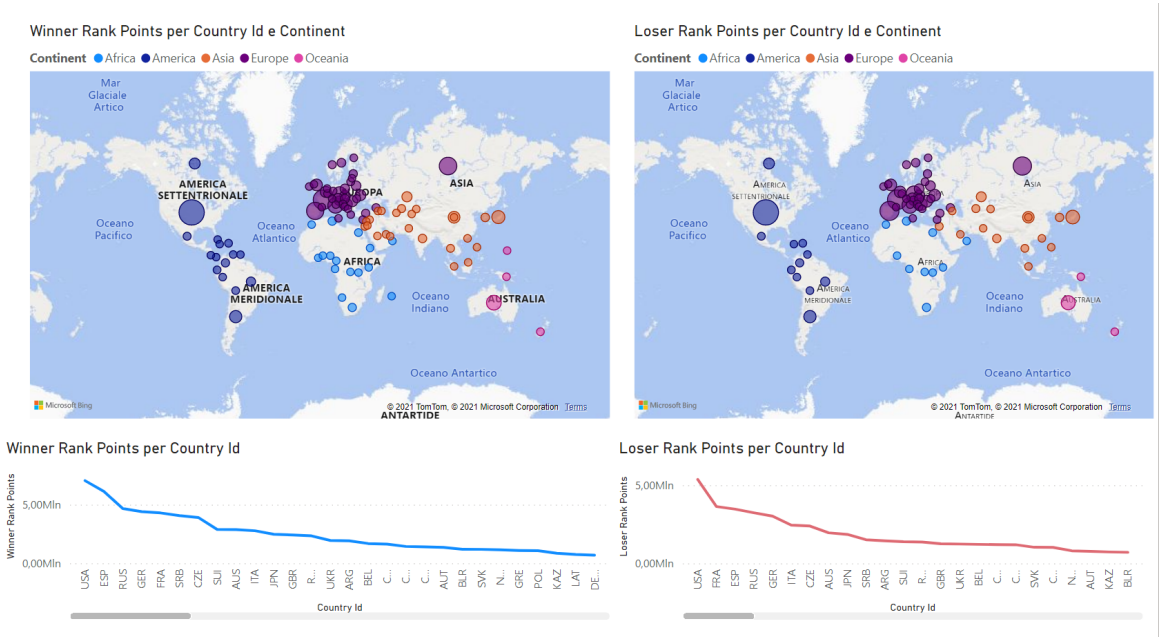

Fig. 14: Query 3 - MDX

3.3.3 Assignment 2: Free-choice dashboard

We decided to produce three different analysis through dashboards on the Group 28 Cube. In Figure 15 we plotted the Winner and Loser Rank Points by Year and Sex. We noticed that for the winners, in 2016 and 2020 the highest rank points were obtained by male players, while female players dominates all the remaining years. Analyzing the Loser Rank Points instead, we can say that out of the players who lost, females had the highest rank points throughout all the years except in 2020.

Fig. 15: Winner and Loser rank points by Year and Sex.

We then wanted to see which country/continent had the largest number of winners and losers, making also a distinction by sex. We noticed that the majority of winners and losers belonged to the continent Europe. Concerning winners, Russia maintains the record, having 2.022,00 female players in the time period that goes from 2016 to 2021. It is worthy to mention that there is a largest number of female players (for both winners and losers) compared to male players (in each continent and country).

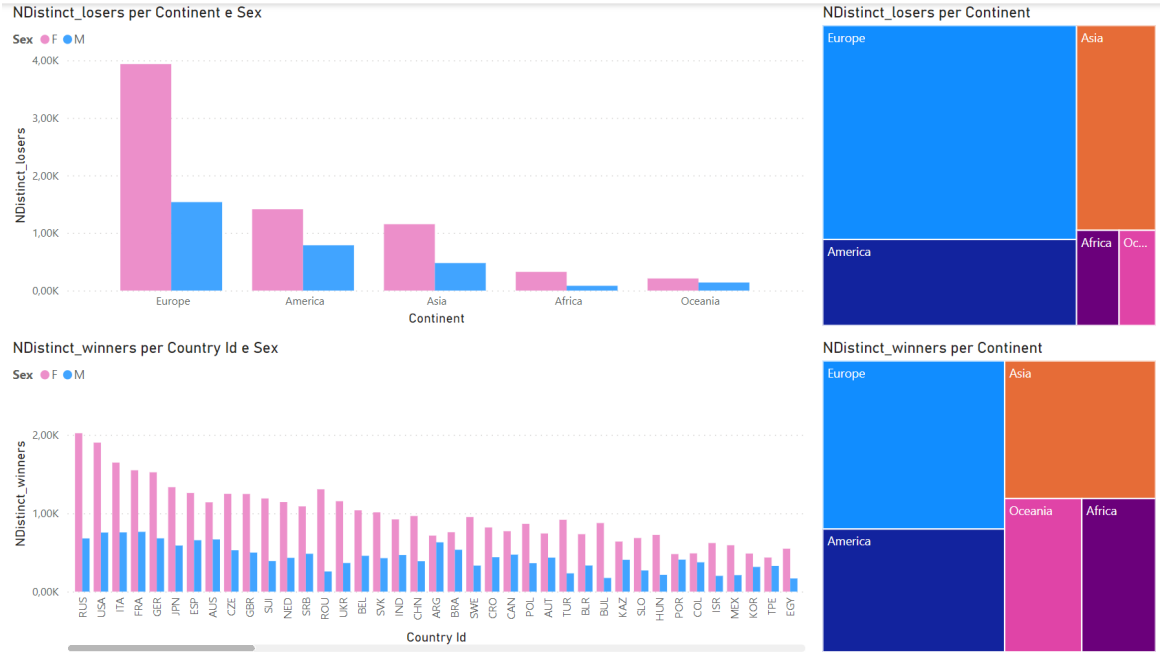

Fig. 16: Number of distinct winners and losers by Continent, Country and Sex.

In our opinion the last dashboard is the most interesting one. We decided to plot the number of spectators and revenue by year, observing the effect that the pandemic period had on tennis competitions. Observing the pie-chart, we can clearly see a drop of spectators in the year 2019 and 2020 (which corresponds to the years when the Covid-19 started spreading). This negative affect can also be seen in terms of revenue, having the total revenue of tennis competitions dropping from $678.551.671,28 in 2019 to $287.271.287,06 in 2021 (almost a drop of 57.6 %). Lastly in the line plot on the right, we plotted the total revenue by tournament, and we established that the most profitable tournament is Wimbledon with a total revenue of $5.002.793,97.

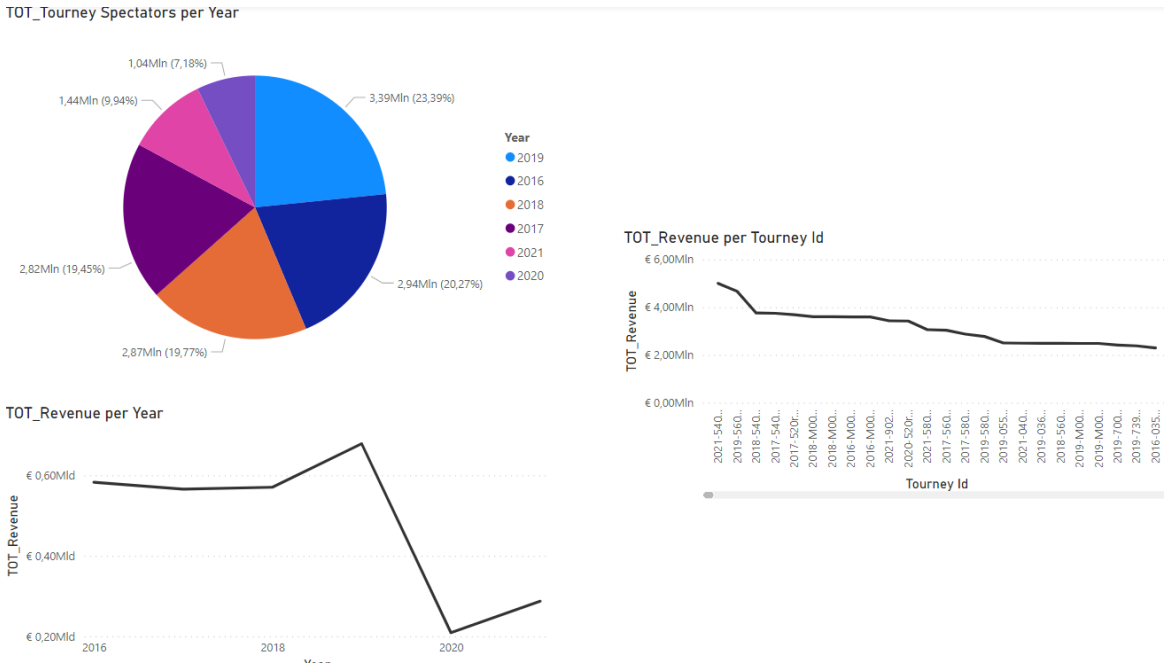

Fig. 17: Total Spectators and Revenue by Year.