INSA Lyon
TC Department

IST-Database and Data Mining

2021/2022 - 1st Semester
November 8th, 2021

DBM1: The 2020 Tokyo Paralympics

Authors
Katharina Alefs, katharina.alefs@insa-lyon.fr, RWTH Aachen University, Germany
João Marques, joao.dos-santos-marques@insa-lyon.fr, IST Instituto Superior Técnico, Portugal

1 Introduction

This report summarizes our project work conducted as part of the *IST-Database and Data Mining* course on concepts concerning the design and the use of databases.

It includes a general presentation of the dataset that the project is based on, insights into our design decisions, the corresponding ER diagram and a run-through of the implementation steps. We will conclude by highlighting problems that we faced during the implementation and how we have solved them.

2 Dataset and Preprocessing

The dataset that we have decided to work with contains data on the 2020 Tokyo Paralympic Games and can be obtained on Kaggle (Data). 4426 Athletes, 212 Teams from 162 Countries participated and competed against each other. The data gives information on the participating athletes and coaches in the Paralympics, the countries that they represent and the events they take part in. Moreover, the `paralympic-medals.csv` file gives information on the teams that won medals.

We made a few adjustments in preprocessing steps to gain new insights as well as make it easier to handle the data. This included splitting the 'Name' column for athletes and coaches into First and Last Name based on capitalization and replacing the 'Female' and 'Male' in the Gender column by 'F' and 'M'.

Upon suggestion, we also added a year column for the medalists to distinguish between the different Paralympic editions and generate more interesting queries. Using `Mockaroo`, we added mock data for the athletes to test these queries.

3 Database Schema and ER Diagram

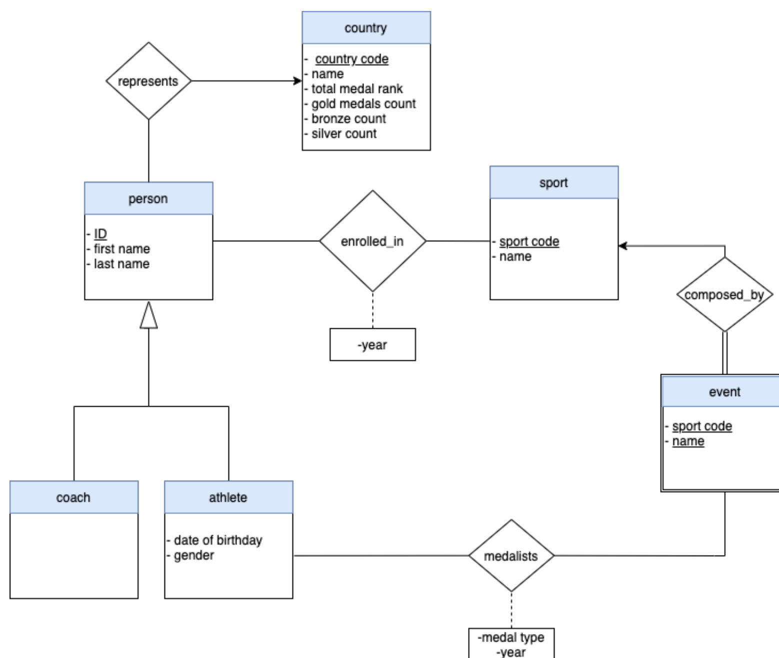

Figure 1: ER Diagram Paralympics

The first entity to be created was the *athlete*, since almost all the information and queries are related to this one. Then, as it has a lot of information in common with the coaches, an entity *person* was created, being the generalization of *coach* and *athlete*. Moreover, an entity *country* was needed to be built in order to gather all the medals each country won. To reduce redundancy of information, each *person* is associated through *represents* with the *country*. An assumption made here to simplify our study case is that each *person* only represents one *country*, so that it could be implemented as one column of *person*.

Since the *athletes* are enrolled in *sports*, the association between both entities seemed logic and the chosen cardinalities enable a player to enroll in several sports during several sessions (thanks to the year attribute). Each *sport* represents a discipline, and each one of those have *events*. We considered this as a weak entity since an event should not exist without a sport.

Finally, the *medalists* were the hardest to model, but each instance of it represents a relation between an *athlete* and an *event* (e.g. "Michael Phelps (athlete) won 100M Man swimming (event)"). It was "cleaner" to model it that way than as part of the event entity, since an event can be won by a team. The medal type and the year attributes enable distinguishing the type of medal won and the time of victory.

4 Implementation of the Database

With the data preprocessed and the ER-Diagram set, the PostgreSQL implementation started.

At first, all the entity tables were created, assigning the respective attributes. After that, it's primary keys were identified and stated according to the ER-Diagram. From that, we set some constraints: athletes and coaches need to exist in the person relation, each person's country has to exist in the country table, all the events have to be associated with a sport (those three are all foreign key constraints) and the gender of an athlete must match a certain format (check constraint).

Finally, we created the tables for the associations. The *represents*, as it inherits a one to many relationship, is just a variable in the person table with a Foreign Key constraint to the country table. The enrolled relationship is a table with the primary keys of person and sport and with a year variable (these three together form a composed primary key to this table). At last, the medalists table has the athlete *id*, the event *name* and $sport_code$ (since they both are needed to identify the event uniquely), the $medal_type$ of the winner and the *year* he/she won. The majority of the constraints are foreign key constraints similar to the previous ones but we implemented an extra constraint to check if the set of attributes ($id, sport_code, year$ match the enrolled table (since an athlete needs to be enrolled in order to win an event). We assume here that each athlete can not win more than one medal for the same event in the same year.

The biggest problem we faced during this process was that the *athlete* and *coaches* instances did not have an ID in the .csv file at hand. By that, we set a DEFAULT insertion in the ID of the table where if no ID is given, it is set sequentially and automatically.

5 Populating the Database

After building all the tables in PostgreSQL, as the data was already preprocessed and the majority of problems addressed, we just needed to write some python scripts in

order to fill the database. For this purpose, we used the library *psycopg*2.

After building them, the order in which we run the scripts to populate is really important since we need to first populate the tables that do not rely on others (e.g. the country table and the sports table) and then all the ones that reference the ones already inserted and so one and so forth. By that, the insertion order was: country, sports,person, athlete, coach,enrolled, events and medalists. In all these python files we check the tables before doing some operation in order to detect if we are inserting something new, adding more to the already added person (e.g. a player that is enrolled in several sports) and prevent duplicates and redundancy. Apart from the python scripts, we wrote a sql file with some mock data insertions in order to accomplish results in queries 7 and 8 (see section below), which the professor added to our project.

While populating our database, we realized that splitting the person's names into first and last name based on capitalization was a big mistake and probably should have been reversed immediately. We sticked with it though and tried to treat edge cases accordingly (as for example last names like McGREGOR were giving us a hard time). Another problem faced during the insertion was ensuring the correct order of transactions and choosing the right order of insertion (e.g. an athlete first has to be inserted in the person table and than in the athlete one). Another problem we took care of was that there was no table for events prior. Because of that, while checking the *medalists.csv* we checked if the event existed already in the database, otherwise it was needed to be added before adding the medalists.

Seemingly smaller problems such as random spaces in the middle of the input data took an unproportional amount of time to fix.

The repository with the code used can be found here.

6 SQL Queries

The questions we wanted to answer with our database were:

1. Number of athletes that where enrolled per discipline?
2. Which first name was the 2nd most popular among the athletes?
3. How many gold, silver and bronze medals did each country win? (medal tally)
4. For each year (1980 - 1990), how many athletes were born then and what are their names (sorted alphabetically)?
5. The birth year of athletes that won exactly 1 gold and 1 silver and 1 bronze medal?
6. From all the winning teams across all events, which team has the most members? What are their names?
7. Is there a female athlete that won a medal in two different disciplines in two different editions?
8. Is there any athlete that skipped one edition but then performed better then his/her former participation?

This section was harder than expected since some queries seemed easier to write before we had the data at our hands. Nevertheless, all of them were achieved, having set some assumptions first:

- The query 4 was split in two queries;
- In query 8, we considered that "improving performance" is not winning a medal at first and than winning a medal. Otherwise the query would be too big because we would have to check improvement for each medal type (but this could be done with UNION clause).

There were no big problems during this section after the assumptions were made but some of the queries took us longer than predicted. The most difficult query to build was probably the last one since it involves a lot of tables and operations at the same time while other big queries like the 4th are almost two same queries put together. The SQL implementation of our queries can be found in the `Queries.sql` file.

7 Advanced Features and Fullstack Development

We added indexes to our database in order to verify wehther they would improveme performance. The indexes we used were the B-Tree index(for query 4 - on the date_of_birthday) and the Hash index (for the 2nd - on first_name in the get first_name - and 8th - on year - queries). The code to insert the indexes can be found in the file `Queries.sql` as well).

To test the performance, we computed the average of running the query 10 times with and without an index. The results can be found in the following table:

Type	Query	Avg.Time Without Index [ms]	Avg.Time With Index [ms]
B-Tree	4	81,25	66,25
Hash	2	174,125	169,778
Hash	8	326,125	327,35

Table 1: Index performance comparison.

We were not expecting results since we do not have data in high quantities, but we achieved some speed improvement, especially on the B-Tree index (which makes sense since we are selecting an interval of values). Nevertheless, the response time for query 8 increased even when adding the index to the tables *medalists* and *enrolled* (the result shown is the best obtained - with 2 indexes).

Probably, if we had millions of records and the selection was really specific (only few records being selected), there the indexes would make a huge difference.

Regarding transactions, we used them a lot during the insertion process in the scripts. Before inserting something, we always checked if there was already a similar insertion in the table. When getting the id of the person inserted to insert in the table athletes, we had to make sure the transaction before already happened in order to correctly use the id for the next table.

We created also a website to visualize the results of the queries:

- https://web.tecnico.ulisboa.pt/ist190114/main.cgi

In here, Python was used as a CGI. The same database used on the project was built through a server available for students from the university "Instituto Superior Técnico" and populated accordingly.

8 Conclusion

We managed to set up a database for the Tokyo Paralympics and implemented Queries in SQL to answer questions concerning the data at hand. We faced certain complications which in the end gave us a deeper understanding of best-practices and our data. In hindsight we could have chosen the queries more strategically, since just making them very complex did not lead to versatile queries. The process was a lot of fun and we are looking forward to explore our data set further in the upcoming Data Mining project.