FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Faculty of Sciences of the University of Porto

Department of Computer Science

Advanced Topics in Databases

Practical Assignment

Aníbal Silva Rui Fernandes

June, 2022

Abstract

This report describes the practical assignment of the Advanced Topics in Databases course.

This practical assignment consists in creating a data warehouse and conducting data analysis on it, as well as creating graphical reports using the Python library `matplotlib`.

In this report, we briefly describe our approach to the problem and discuss the decisions we made.

Contents

List of Figures

1 Introduction

The aim of this practical assignment consists in the construction of a datawarehouse and the elaboration of data analysis over it, with the construction of graphical reports using the Python module `matplotlib`.

The data warehouse contains data from national swimming competitions at the master level (*i.e.* class of competitive swimming for swimmers 25 years and older), namely Troféu Pescada 2021 and the Summer 2021 Championship.

The data is imported into the datawarehouse after being extracted from files in the Lenex (LXF) format. Some graphical reports will then be generated based on this data.

Structure of the Report

The remainder of the report is structured as follows:

- In Section 2, **Data Model**, we describe the data contained in the data warehouse.
- In Section 3, **Data Analysis & Visualization**, we provide some insight into the data by looking into relevant statistics as well as plotting the data we considered relevant.
- Finally, Section 4, **Conclusions & Future Work**, concludes the report and suggests remarks for future work.

2 Data Model

In this section, we describe the data tables contained in the data warehouse.

Some modifications were made in the original script. This was mainly motivated due to the fact that same athletes and clubs had different `ids` for different meets, when in our perspective they should be uniquely identified across tournaments. They were defined as:

- `athleteid` = `firstname` + `lastname` + `birthdate` + `inc_id`, where the `inc_id` increments when two athletes share the same first and last name and the birthdate;
- `clubid` = `code` + `nation` + `region`;
- `resultid` = `meetid` + `resultid`;
- `swimstyleid` = `distance` + `relaycount` + `stroke`;
- `eventid` = `meetid` + `eventid`;
- The `license`, which originally was `meetid` + `clubid` + `idx` was replaced by `athleteid` + `meetid` because the same athlete could have changed between teams for different tournaments.

Regarding our data model, we built two fact tables, one that gathers information regarding a club in a tournament, while another gathered information of a given athlete in a tournament. Our schema is illustrated in Figure 1.

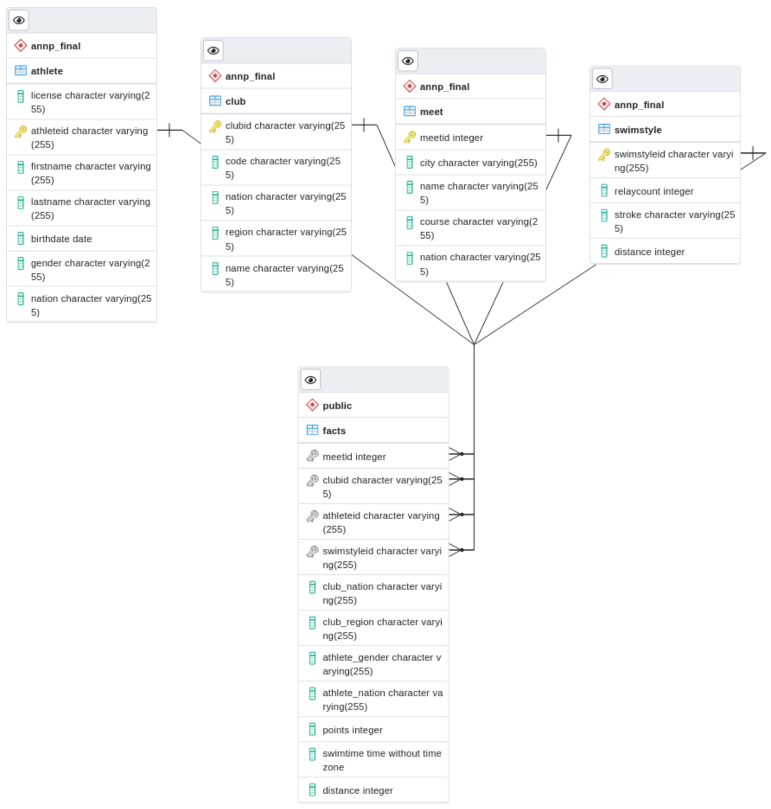
annp_final
athlete
license character varying(255)
athleteid character varying (255)
firstname character varying (255)
lastname character varying (255)
birthdate date
gender character varying(255)
nation character varying(255)
annp_final
club
clubid character varying(255)
code character varying(255)
nation character varying(255)
region character varying(255)
name character varying(255)
annp_final
meet
meetid integer
city character varying(255)
name character varying(255)
course character varying(255)
nation character varying(255)
annp_final
swimstyle
swimstyleid character varying(255)
relaycount integer
stroke character varying(255)
distance integer
public
facts
meetid integer
clubid character varying(255)
athleteid character varying (255)
swimstyleid character varying(255)
club_nation character varying(255)
club_region character varying(255)
athlete_gender character varying(255)
athlete_nation character varying(255)
points integer
swimtime time without time zone
distance integer

Figure 1: Datawarehouse schema

This Figure shows two fact tables, one related to overall statistics of a team in a tournament, while the other containing the overall statistics of an athlete. Regarding the first table, *Club de factos*, we grouped our data by `meetid`, `clubid` and `swimstyleid` to extract the following statistics:

- `number_of_players` – The number of players;
- `average_age` – The average age of players;
- `total_points` – The total points a given team had in a tournament;
- `total_distance` – The total distance all the players swam in a team;
- `average_swimtime` – The average swimtime all the players swam.

Regarding the second fact table, *Athletes de factos*, we grouped our data by `athleteid`, `meetid` and `from_club` in order to extract the following statistics:

- `average_swimtime` – The average swimtime a given athlete swam

  - `average_points` – The average points an athlete had
  - `average_distance` – The average distance an athlete swam

for a given tournament.

3 Data Analysis & Visualization

In this section, we build informative reports that analytically summarize the information stored in the datawarehouse. These reports are also be presented in graphical form when applicable.

3.1 Athletes by Age

To determine to determine the average age of the athletes, we can run the following SQL query:

```
SELECT AVG(age(birthdate))
FROM annp_final.athlete;
```

From this, we can see that the average age of the athletes is 46 years, 6 months and 31 days.

We can also determine who's the youngest athlete by running the following SQL query:

```
SELECT *
FROM annp_final.athlete
ORDER BY age(birthdate) ASC
LIMIT 1;
```

– **Name:** Ana Mónica Eloi

– **Gender:** Female

– **Birthdate:** 29/12/1996

– **Age:** 25 years

On the other hand, we can learn information about the oldest athlete by running the following SQL query:

```
SELECT *
FROM annp_final.athlete
ORDER BY age(birthdate) DESC
LIMIT 1;
```

– **Name:** Virgílio Zacarias Costa

– **Gender:** Male

– **Birthdate:** 21/07/1931

– **Age:** 90 years

Finally, to determine the number of athletes by age, we can run the following SQL query using the PostgreSQL's built-in `age` function:

```
SELECT
    COUNT(*),
    EXTRACT(YEAR FROM age(birthdate)) AS age
FROM annp_final.athlete
GROUP BY age
ORDER BY age ASC;
```

We can then plot the result, as illustrated in Figure 2.

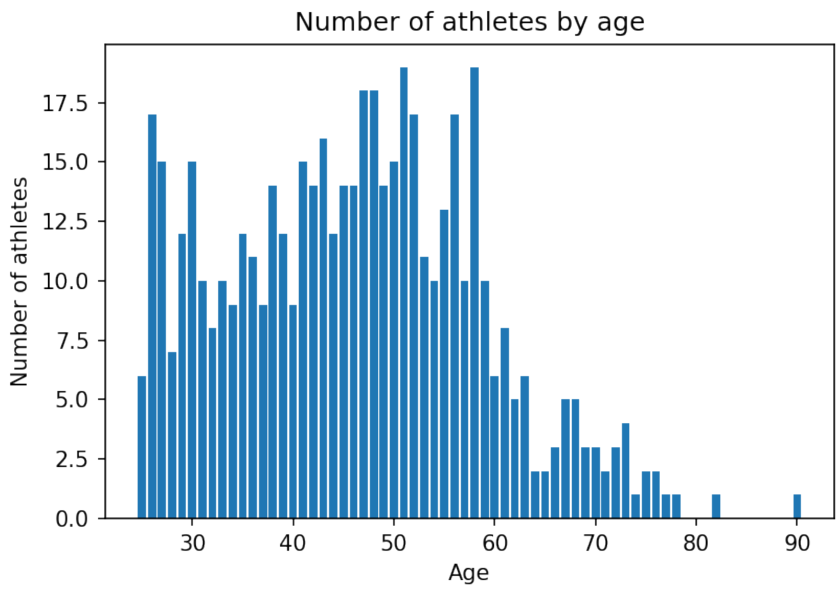

Figure 2: Number of athletes by age

We can also plot the number of athletes by age based on their gender, as illustrated in Figure 3.

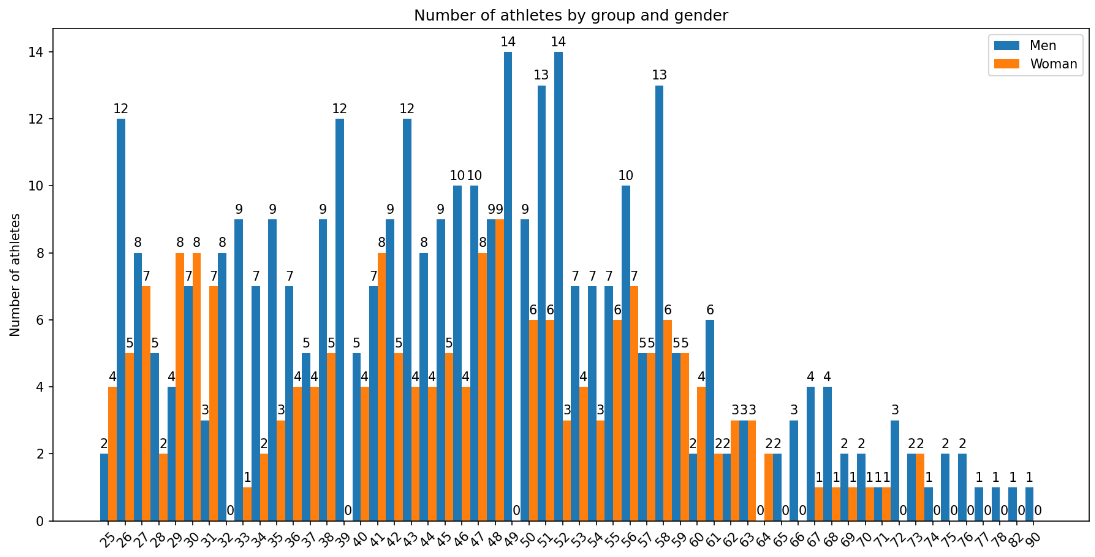

Figure 3: Number of athletes by age and gender

3.2 Athletes by Nation

To determine the number of athletes by nation, we can run the following SQL query:

```
SELECT
    nation,
    COUNT(*) AS nationCount
FROM annp_final.athlete
GROUP BY nation
ORDER BY nationCount ASC;
```

We can then plot the result, as illustrated in Figure 4.

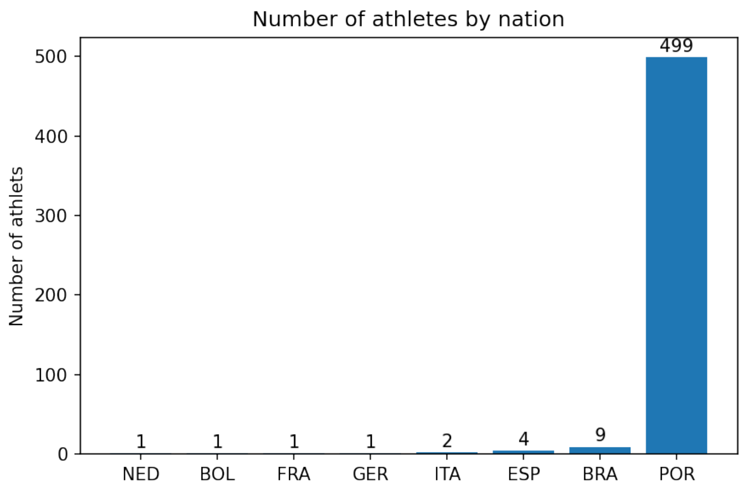

Figure 4: Number of athletes by nation

We can also take the gender into account with following SQL query:

```
SELECT
    nation,
    gender,
    COUNT(*)
FROM annp_final.athlete
GROUP BY CUBE (nation, gender)
ORDER BY nation, gender NULLS LAST;
```

Plotting the result, we have the following:

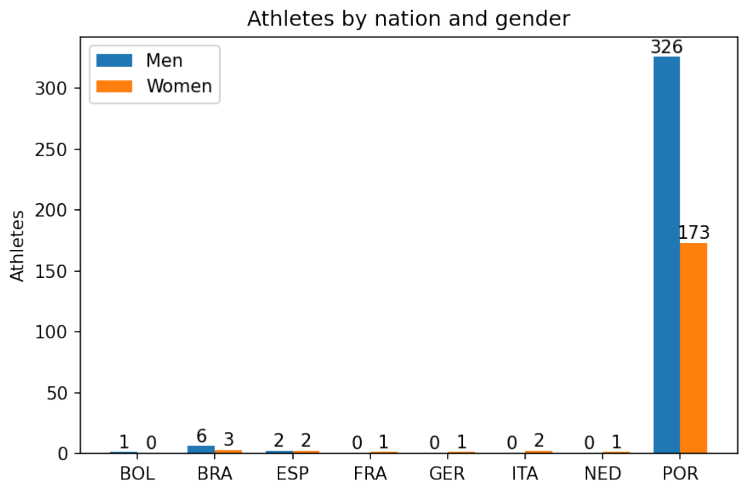

Figure 5: Number of athletes by nation and gender

3.3 Athletes by Gender

To determine the number of athletes by gender, we can run the following SQL query:

```sql
SELECT
    gender,
    COUNT(*)
FROM annp_final.athlete
GROUP BY gender;
```

We can then plot the result, as illustrated in Figure 6.

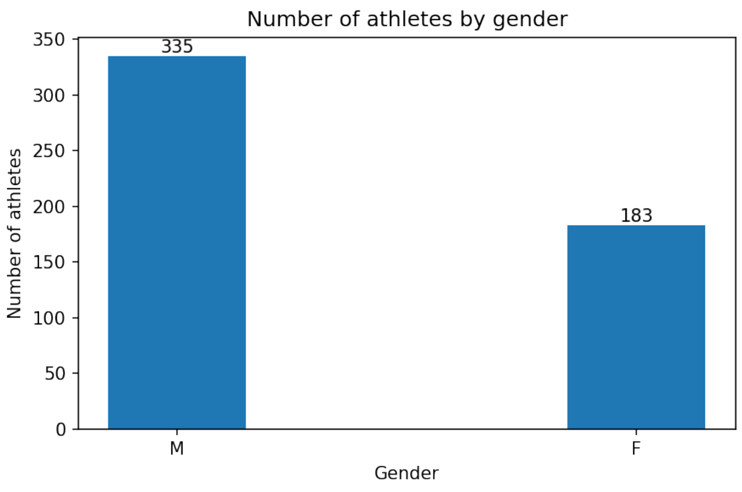

Figure 6: Number of athletes by gender

We can also plot this in a pie chart, as illustrated in Figure 7.

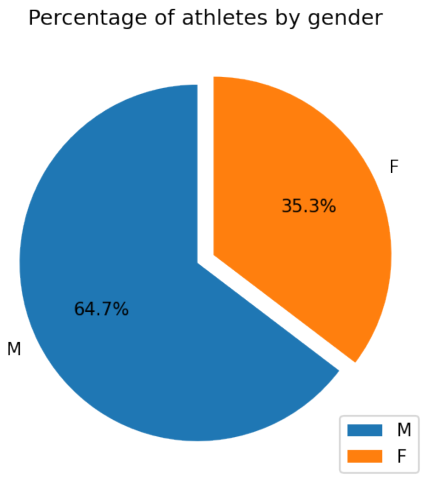

Figure 7: Percentage of athletes by gender

3.4 Clubs by Nation

We can determine the number of clubs by each nation by running the following SQL query:

```
SELECT
    nation,
    COUNT(*) AS nationCount
FROM annp_final.club
GROUP BY nation
ORDER BY nationCount ASC;
```

Plotting the result in a bar chart, we have the following:

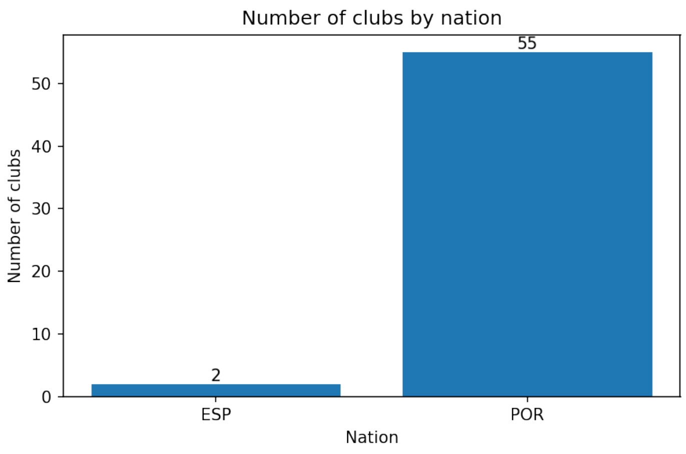

Figure 8: Number of clubs by nation

We can also plot this in a pie chart, which gives us the following:

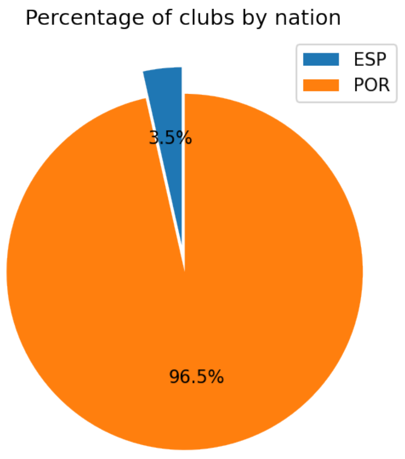

Figure 9: Percentage of clubs by nation

3.5 Clubs by Region

To determine the number of clubs per each region in Portugal, we can run the following SQL query:

```
SELECT
    region,
    COUNT(*) AS regionCount
FROM annp_final.club
WHERE region SIMILAR TO '[A-Z]+'
GROUP BY region
ORDER BY regionCount ASC;
```

Plotting the result of the query in a bar chart we have the following:

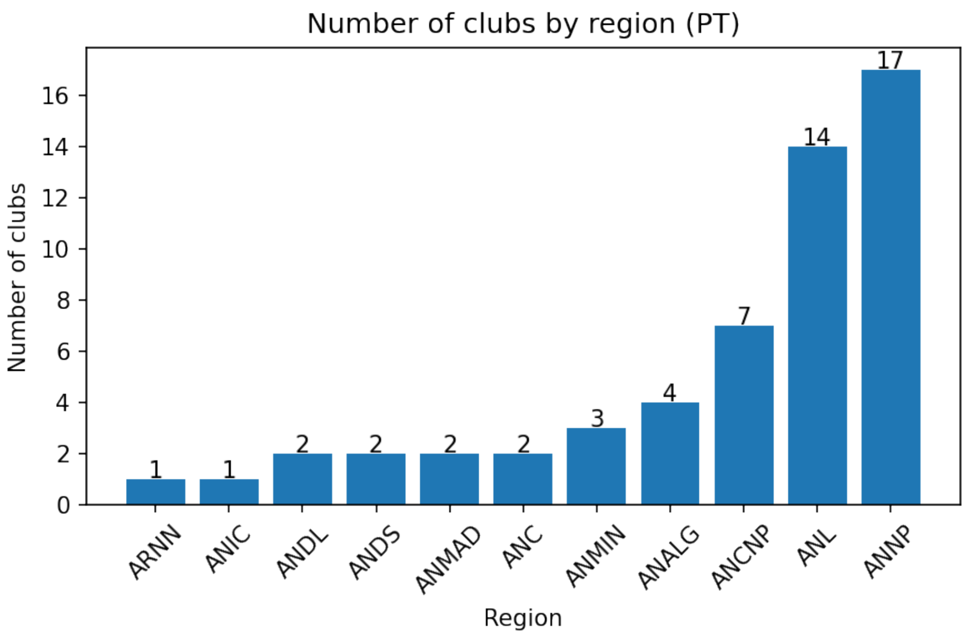

Figure 10: Number of clubs by region

Here, it is worth noting that this query only considers the portuguese clubs, because each is identified by uppercase letters only. Spanish regions, on the other hand, are identified with numbers, which means that if we want to run the previous query considering only spanish regions, we only have to replace `WHERE region SIMILAR TO '[A-Z]+'` with `WHERE region SIMILAR TO '[0-9]+'`, as shown bellow:

```
SELECT
    region,
    COUNT(*) AS regionCount
FROM annp_final.club
WHERE region SIMILAR TO '[0-9]+'
GROUP BY region
ORDER BY regionCount ASC;
```

However, this has a downside in the sense that we have no way of knowing which regions these values 10114 and 11115 refer to. Ploting the result in a bar plot, we have the following:

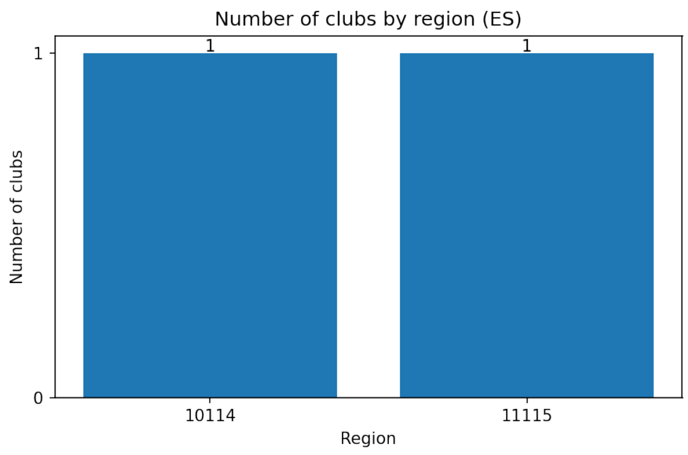

Figure 11: Number of clubs by region (ES)

We can also plot this a pie chart, which gives us the following:

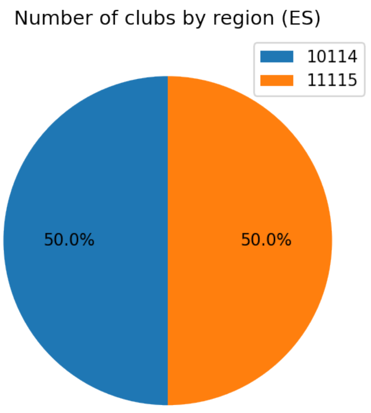

Figure 12: Number of clubs by region (ES)

3.6 Club facts statistics

3.6.1 Overall Statistics

```
    SELECT
    CASE GROUPING(af.clubid)
        WHEN 1 THEN 'all_clubs'
        ELSE af.clubid
    END AS "Club",
    CASE GROUPING(af.meetid)
        WHEN 1 THEN 'all_meets'
        ELSE af.meetid
    END AS "Tournament",
    ROUND(AVG(points), 2) AS "Average Points",
    ROUND(AVG(distance), 2) AS "Average Distance",
    ROUND(AVG(swimtime), 2) AS "Average Swimtime",
    ROUND(AVG(birthdate)) AS "Average Age",
    COUNT(af.athleteid) AS "Total Athletes"
FROM (
    SELECT
        CAST(meetid AS VARCHAR(255)),
        f.athleteid,
        clubid,
        points,
        distance,
        ROUND(CAST(
              (EXTRACT(
                 MICROSECONDS FROM swimtime))/10^6
                        AS NUMERIC), 2) AS swimtime,
        round(EXTRACT(
            EPOCH FROM
                CURRENT_TIMESTAMP - a.birthdate)/(365*24*60*60)) AS birthdate
    FROM facts f
    JOIN annp_final.athlete a ON a.athleteid = f.athleteid
    ) af
WHERE points IS NOT NULL
GROUP BY CUBE (af.clubid, af.meetid)
ORDER BY "Average Points" DESC;
```

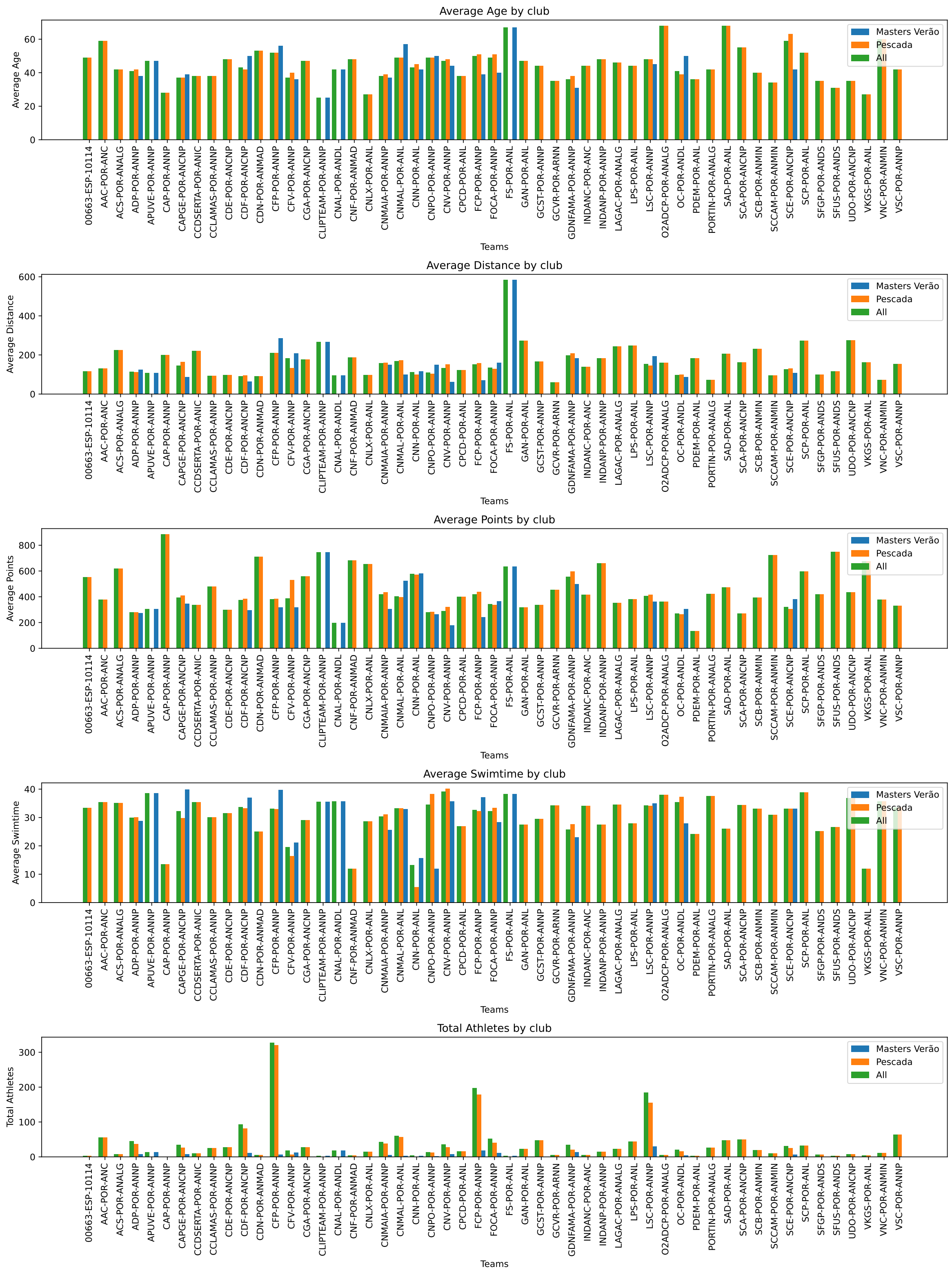

Figure 13: Statistics from fact Club table.

3.6.2 Team & Swim Style Statistics

Next, we also show the overall statistics given a swim style. For that, we use the following SQL query:

```
SELECT
    CASE GROUPING(c.code)
        WHEN 1 THEN 'all_clubs'
        ELSE c.code
    END AS "Team",
    CASE GROUPING(af.swimstyleid)
        WHEN 1 THEN 'all_styles'
        ELSE af.swimstyleid
    END AS "SwimStyle",
    ROUND(CAST(
                AVG(EXTRACT(
                        MICROSECONDS FROM af.swimtime))/10^6
                            AS NUMERIC), 2) as "Average Swimtime",
    round(AVG(af.points), 2) AS "Average Points",
    COUNT(af.athleteid) AS "Total Athletes"
    FROM facts af
    INNER JOIN annp_final.club c ON c.clubid = af.clubid
    WHERE af.points IS NOT NULL
    GROUP BY CUBE (af.swimstyleid, c.code)
```

To filter the amount of information this table has, we filter, for a given swim style, the top 5 teams that had the higher total of points. This is done for all the tournaments. This pre-processing step was done in Python and the result is depicted in Figure 14. Note that not all the bar plots have 5 teams. This is due to the lack of data presented from both tournaments.

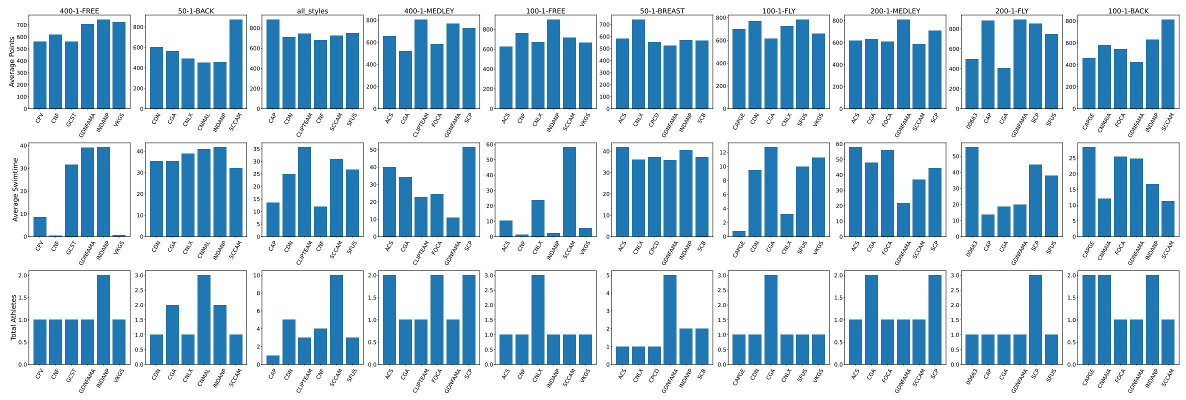

Figure 14: Top 5 teams per swim style for a given statistic.

3.7 Athlete Statistics

3.7.1 Athletes that commited fault

Here, we assume that an athlete commited a fault if it has 0 points in a given swimtime.

```
    WITH tmp AS (SELECT
        athleteid,
        CASE
                WHEN points IS NULL THEN 'fault'
                ELSE 'notfault'
        END AS fault
FROM facts)
SELECT
        athleteid,
        count(tmp.fault) AS "Number of Faults"
FROM tmp
WHERE fault = 'fault'
GROUP BY athleteid
ORDER BY "Number of Faults" DESC;
```

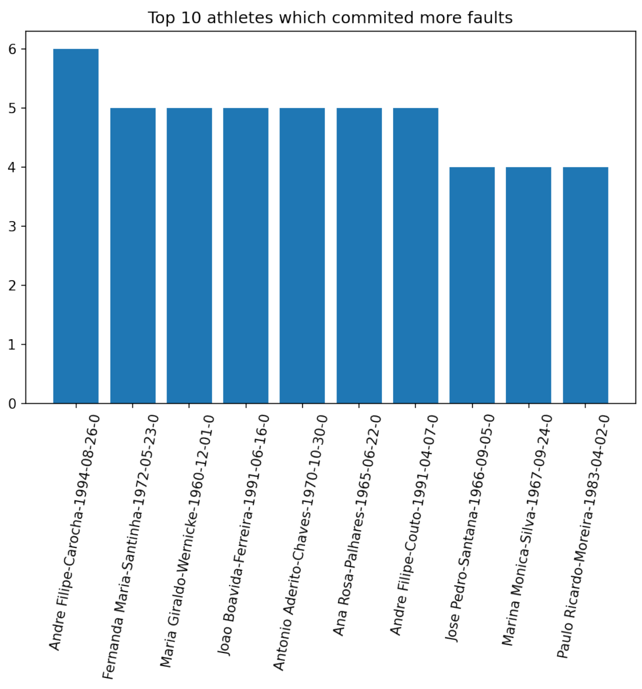

Figure 15: Athletes that commited more faults

3.8 Athlete & Swim Style Statistics

```
    SELECT
    CASE GROUPING(a.firstname)
        WHEN 1 THEN 'all_players'
        ELSE a.firstname
    END AS "Athletes",
    CASE GROUPING(af.swimstyleid)
        WHEN 1 THEN 'All SwimStyles'
        ELSE af.swimstyleid
    END AS "SwimStyle",
    ROUND(AVG(points), 2) AS "Average Points",
    ROUND(AVG(swimtime), 2) AS "Average Swimtime"
FROM (
    SELECT
        athleteid,
        CAST(swimstyleid as VARCHAR(255)),
        points,
        distance,
        ROUND(CAST(
                      (EXTRACT(
                                MICROSECONDS FROM swimtime))/10^6
                               AS NUMERIC), 2) AS swimtime
    FROM facts) af
JOIN annp_final.athlete a ON a.athleteid = af.athleteid
WHERE points IS NOT NULL
GROUP BY CUBE (a.firstname, af.swimstyleid)
ORDER BY "Average Points" DESC;
```

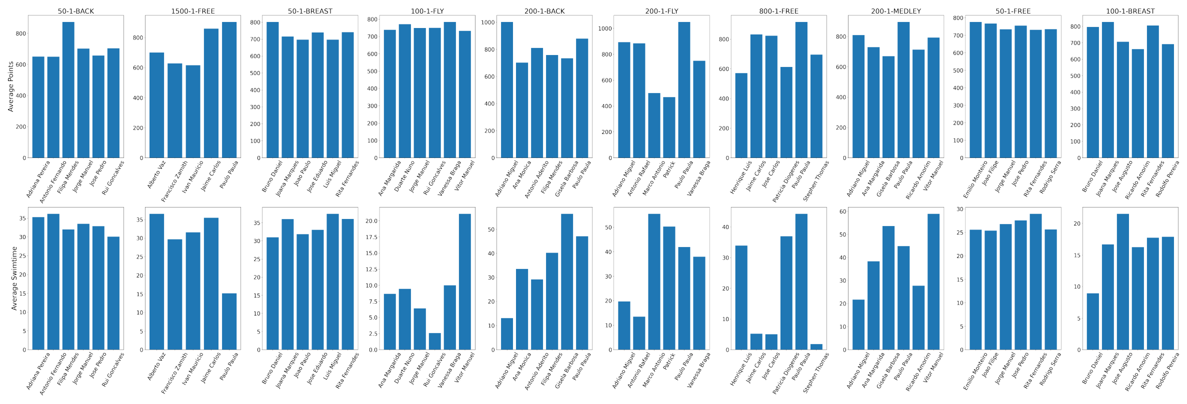

Figure 16: Average points and swim time for a given athlete and a given swim style

4 Conclusions & Future Work

In this practical assignment, we were able to model, create and load the datawarehouse with the extracted information from files in the Lenex (LXF) format, as well as build informative reports that analytically summarize the information stored in the datawarehouse.

It is also worth mentioning that, after restructuring the datawarehouse, we were no longer able to query how many athletes have participated in both tournaments.

As future work, it would be important to include information about other tournaments in the database. To do this, we would have to pre-process the data to ensure it conforms to the database schema.

Finally, it would also be important to explore other SQL operators such as `GROUP BY ROLLUP`.