Advanced Databases Report

Gonçalo Lopes 56334
Frederico Prazeres 56269
Ricardo Sobral 56332
10/12/2023

Ciências
ULisboa

1. Database Description

For our dataset we decided to pick the Formula 1 dataset. The dataset consists of storing information about the Formula 1 races, drivers, constructors, results, and some more information from 1950 till the latest 2023 season. We've decided to utilize only the races, drivers, constructors, and results from the database, and we've made some changes that are described below.

2. Database Schemes

2.1 SQL database

We've decided to remove some of the data that was not really all that interesting to us, and that we wouldn't use on our queries. For the SQL database the scheme looks like this:

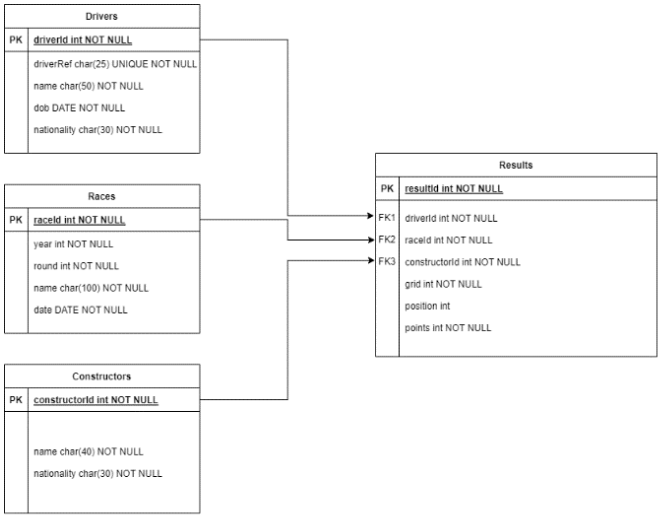

Figure 1 - SQL DB scheme

We've focused primarily on making connections between the results table and the other tables, so the complex queries would be straightforward and easy to manage.

2.2 MongoDB database

Similarly, the MongoDB scheme looks like:

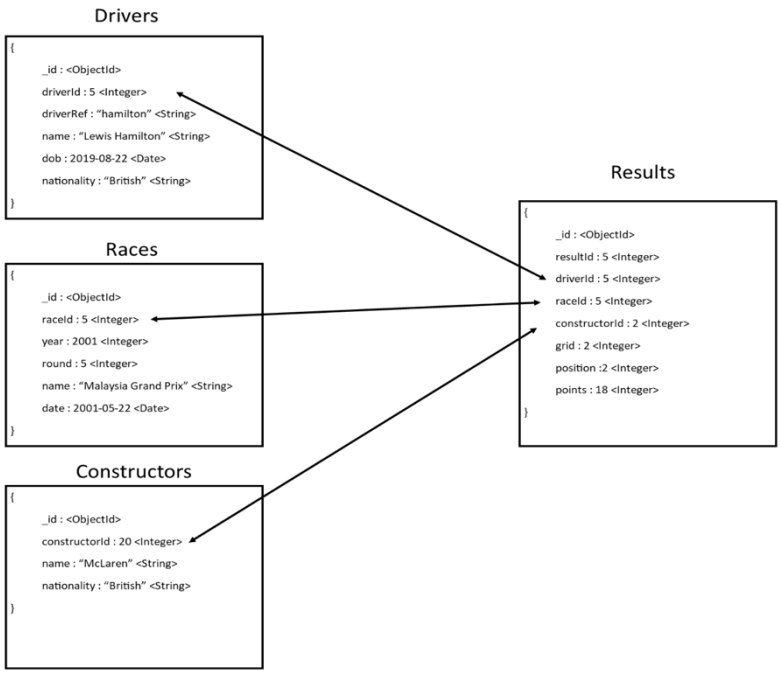

Figure 2 - MongoDB scheme

The only difference between the schemes is that the MongoDB scheme has an entry called *_id* that is the default entry when creating a MongoDB collection.

3. Discussion of points done/not done in the project

The only point that wasn't "done" in the project was point "c.iii", this is because we felt that it was not necessary to join any tables since all our tables only had relation to one table which was the table in common for all of them. Given that, we felt that it was better to maintain the scheme as we originally created it and proceed from there.

3.1 Creating the databases

3.1.1 MongoDB

Firstly, the csv is read to a *pandas Dataframe* variable for each csv, so for the csv *results.csv* there's a corresponding *f_results*, afterwards, we drop the unnecessary columns that we originally planned not to use and make any changes if that's the case. We then convert the data to *json* so it can be loaded into the MongoDB. Then we create a cluster for each of the files so again, for a *f_x* there's a *cluster_x* and we insert the *json* data there.

3.1.2 SQL

For the SQL version, firstly, we create an empty table with the data that we planned on the scheme. We don't need to convert the data to *json*, therefore, we select the columns that we originally planned to drop, read the csv in a *pandas Dataframe*, drop the columns, and insert into the previously created table the values of dataset.

3.2 Queries

3.2.1 Simple Queries

Our two chosen simple queries are:

1. Return the years (sorted) where Monaco Grand Prix occurred.
2. Return the id of the races where the driver with id 3 won (first position)

3.2.1.1 MongoDB

For the first query, this only affects one table which is *races*, therefore we perform a simple instruction of, creating a restriction where the name equals "Monaco Grand Prix", then we sort the column "year", and we print only that column.

```
my_query = {'name': 'Monaco Grand Prix'}

result_my_query = cluster_races.find(my_query).sort([('year', 1)])
```

Figure 3 - First simple query (MongoDB)

For the second query, this also only affects one table which is *results*, we restrict the *driverId* to equal 3 and the position to equal 1 and we return the column *raceId* of the results.

```
my_query = {'$and':[{'driverId':3},{'position':1}]}
result_my_query = cluster_results.find(my_query)
```

Figure 4 - Second simple query (MongoDB)

3.2.1.2 SQL

Similarly, we select that table, equal the name column to "Monaco Grand Prix" and then order by the year ascendingly.

```
mycursor.execute("SELECT year FROM races WHERE name = 'Monaco Grand Prix' ORDER BY year ASC")
```

Figure 5 - First simple query (SQL)

Again, selection of the table results, equal the position to one and the driverId to 3 and print the column raceId.

```
query = """
    SELECT r.raceId
    FROM results r
    WHERE r.driverId = 3 AND r.position = 1
"""
```

Figure 6 - Second simple query (SQL)

3.2.2 Rest of the Queries

Our two chosen complex queries were:

4. Return all the drivers that names start with L and managed to win a race with Mercedes.
5. All the points accumulated by "Max Verstappen" at the Monaco Grand Prix while racing for RedBull.

3.2.2.1 MongoDB

We will not insert any screenshots regarding the MongoDB complex queries since the pipeline that was created would take too much space. With that in mind, it's also worth to note that all these queries were ran using the *Database.command()* so that we could have access to the execution stats of the query and compare them with the non-index version.

For the first query, we created a pipeline, looking up all the tables besides the one we're executing the query in, use regex to match the *driver.name* to begin with L, equal the constructor's name to Mercedes, and the position to 1. We aggregate this pipeline on the *results* cluster, and we print the number of results and the first result obtained.

```
82 results found.
First one:
Lewis Hamilton ended in position number 1 at the 2013 Hungarian Grand Prix for Mercedes
```

Figure 7 - First complex query result (MongoDB)

Like the previous query, we create a pipeline, equal the driver's name to Max Verstappen, the constructor's name to Red Bull (this is because Max also raced for Scuderia Toro Rosso) and the race name to Monaco Grand Prix. The only difference is that we group the points column and sum them up, creating this result:

```
Max Verstappen has accumulated 89.0 points in all Monaco Grand Prix that he has raced for Red Bull.
```

Figure 8 - Second complex query result (MongoDB)

For the Update query, we noticed that there was a "Portuguese Grand Prix" that happened in the year 1958, therefore our update is, on the first execution, to change that year to 2023, and on future executions if the year is 2023, we change it back to 1958 so we can perform it in a cycle, so on the first execution it looks like:

```
UPDATE OPERATION

Before the update, query result:


After the update, query result:
{'_id': ObjectId('65739b80e19823cc42b9a68b'),
 'date': '1958-08-24',
 'name': 'Portuguese Grand Prix',
 'raceId': 773,
 'round': 9,
 'year': 2023}
```

Figure 9 - Update query (MongoDB)

For the Insert query, we created a new driver named "Goncalo Miguel Prazeres", a new constructor named "FCUL" a new race which was a "Portuguese Grand Prix" in 2024 and a new result which simulated that the new driver won the race while racing for FCUL, inserted them in their respective clusters and performed a complex query just to show that the insertion succeeded:

```
After the insertion, query result:
1 results found. First one:
Goncalo Miguel Prazeres ended in position number 1 at the 2024 Portuguese Grand Prix for FCUL starting at grid 20
```

Figure 10 - Insertion query result (MongoDB)

3.2.2.2 SQL

Same complex query as above, we want the drivers whose name start with L that won races for Mercedes; therefore, we must join the tables and connect them by the id that we gave them during the scheme:

```
SELECT COUNT(*) AS driver_count
FROM drivers d
JOIN results r ON d.driverId = r.driverId
JOIN races ra ON r.raceId = ra.raceId
JOIN constructors c ON r.constructorId = c.constructorId
WHERE d.name LIKE 'L%'
  AND c.name = 'Mercedes'
  AND r.position = 1
```

Figure 11 – First Complex Query (SQL)

And we obtain the same result as the query in MongoDB.

Same scenario as above regarding joining the tables and connecting them by the id but instead of performing a count instead we perform a sum, since it is the accumulation of points.

```
query = """
    SELECT SUM(r.points) AS total_points
    FROM drivers d
    JOIN results r ON d.driverId = r.driverId
    JOIN races ra ON r.raceId = ra.raceId
    JOIN constructors c ON r.constructorId = c.constructorId
    WHERE d.name = 'Max Verstappen'
      AND ra.name = 'Monaco Grand Prix'
      AND c.name = 'Red Bull'
"""
```

Figure 11 – Second Complex Query (SQL)

Regarding the Update query, the scenario is the same, where we update the year to 2023 if it's 1958 or vice-versa. Returning this:

```
Race year updated to 1958.
```

OR

```
Race year updated to 2023.
```

Figure 12 & 13 – Returned text of the update operation (SQL)

Although for the insert we decided to only insert a single race equal to the one in MongoDB.

```
(1141, 2024, 1, 'Portuguese Grand Prix', datetime.date(2024, 1, 10))
```

Figure 14 – Returned text of the insert operation (SQL)

3.3 Indexing

As mentioned beforehand we used explain on every query, although we'll only showcase improvements on the report of one simple query and one complex due to the fact of page limitation.

3.3.1 MongoDB

For the simples queries we used the method explain, to showcase the number of documents examined by the query, the number of results (NReturned) and time to complete the query.

```
Explanation for the first query without indexes:
Total docs examined = 1101
NReturned = 69
Total time of the query in milliseconds = 12


First query result with indexes:

Explanation for the first query with indexs:
Total docs examined = 69
NReturned = 69
Total time of the query in milliseconds = 1
```

As we can see the number of documents examined, using indexes, decreased drastically and the same happens for the time consumed, proving the importance of using indexes even for small queries.

Figure 15 – Comparison results of the first simple query with and without indexes (MongoDB)

Due to the limitations of the explain method when using pipelines, which we used for the complex queries, we've decided to only take into account the time spent in each query. Another thing we had to consider was irregularities

of the time it took for each query to run, for some reason, in some runs the time spent by the query without indexes was inferior comparing to the same query using indexes. In order to solve this minor issue, we've decided to run each complex query 10 times (with and without indexes) and print the mean total time of those runs.

```
Mean Total time in milliseconds for the second complex query = 114.9
Mean Total time in milliseconds for the second complex query with indexes = 113.3
```

Figure 16 – Comparison results of the second complex query with and without indexes

3.3.2 SQL

On the second simple query we created indexes based on the driverId and his position which was what we required from the query, and we can see improvements:

```
Races ids where driver with Id 3 (Nico Roseberg) won:
--------------------------------No Indexes--------------------------------


Total time taken without indexes (SECOND SIMPLE QUERY) 0.0009996891021728516
Number of Files Examined  26289


(1, 'SIMPLE', 'r', None, 'ALL', None, None, None, None, 26289, 1.0, 'Using where')

--------------------------------With Indexes--------------------------------


Total time taken with indexes (SECOND SIMPLE QUERY): 0.0
Number of Files Examined  23


(1, 'SIMPLE', 'r', None, 'ref', 'index_result3', 'index_result3', '9', 'const,const', 23, 100.0, None)
```

Figure 17 – Comparison results of the second simple query with and without indexes (SQL)

Especially since we decreased the number of files examined by over a thousand times.

Regarding the second complex query, we created indexes based on every foreign key because of the connections to other tables and the points since that was the objective to accumulate.

We're going to type the results due to page limitations.

- Without indexes
  - Time = 0,0004s
  - Files Examined =26080+1+1
- With indexes
  - Time = 0,0003s
  - Files Examined =857+30+1

Again, very few files examined compared to the no index version, which is a good showcase of improvement.

4. How to replicate the project

There are a few files, in this section we will explain what each file is for.

First, please verify that you have a mySQL instance running in your computer and modify the password section in all SQL related files (there's SQL in the name). Every file is executed simply by running "py x.py" in your console.

There are two create files, one for each database, the "createMongoDB.py" file resets by its own so if you run it again it will delete the previous database and create a new one, if you would like to recreate the SQL database, please run the "dropSQLDB.py" file first.

Afterwards there are two files related to queries. The "queriesMongo.py" file already has indexes integrated, so you can verify the results of the queries, and the time it takes to run them with and without indexes. However, for the SQL version, it only runs the queries and shows the results from executing them. If you would like to verify the indexing performance of SQL, please run "indexSQL.py".