Project for Advanced Databases Course

Master degree in Data Science – 2022/2023

Faculty of Sciences of University of Lisbon

Ana Araújo
Nº 59457

1. Description of the dataset

For this project, the dataset consists of all information on the Formula 1 races, drivers, constructors, qualifying, circuits, lap times, pit stops, championships from 1950 till the latest 2021 season. The data set was taken from: https://www.kaggle.com/datasets/rohanrao/formula-1-world-championship-1950-2020)

Form the Dataset in Dataset Formula1, it was considered:

- Constructors - in F1 the teams are called constructors, and this present their information such as name and nationality
- Constructor Standing - This table shows what is the standing of each constructor after each race.
- Races - This table connects many dots for retrieving information. It contains the year in which race was held, what was the sequence of the races in that year and at what circuit race event occurred.

For this project, it was chosen the data from constructors, constructors results and Races, to verify which teams present more wins in one race and which year that happen.

2. Scheme for both databases with the data described above

Schema for RDMS database:

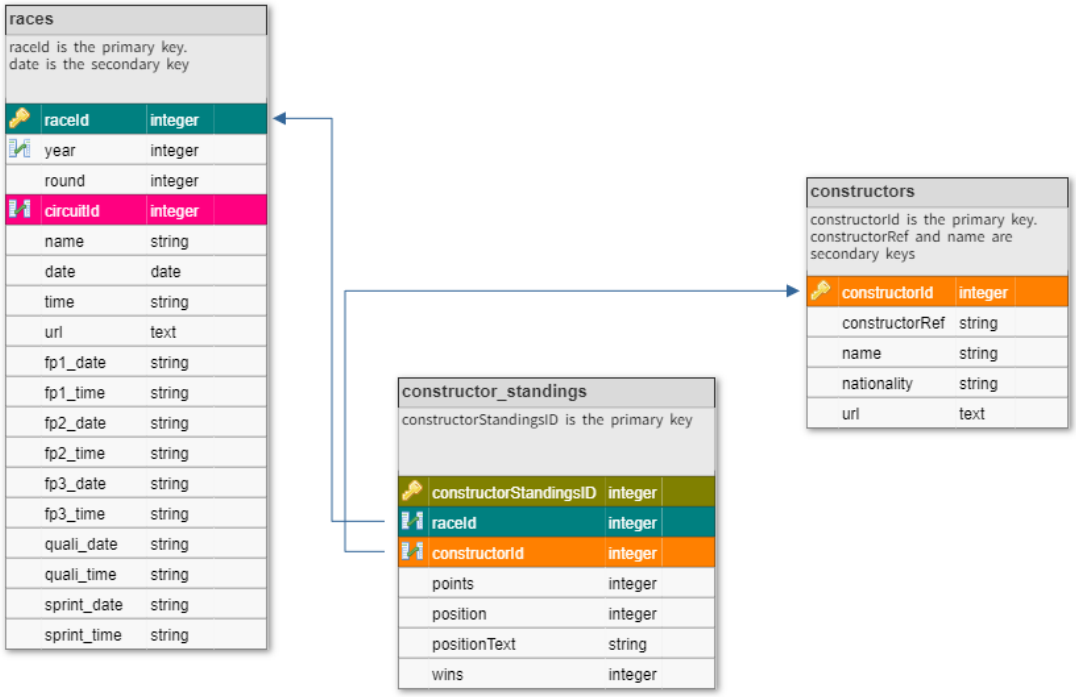

Figure 1. RDMS schema for Formula1 dataset

Schema for noSQL:

Constructor_standings	**Data type**
_id	ObjectId
constructorStandingsId	ObjectId
points	integer
position	integer
positionText	string
wins	integer

constructors	**Data type**
_id	ObjectId
constructorId	ObjectId
constructorRef	string
name	string
nationality	string
url	string

Races	**Data type**
_id	ObjectId
raceId	ObjectId
year	integer
round	string
circuitId	ObjectId
name	string
date	Date
time	Time
url	string
fp1_date	string
fp1_time	string
fp2_date	string
fp2_time	string
fp3_date	string
fp3_time	string
quali_date	string
quali_time	string
spring_date	string
spring_time	string

Figure 2. NoSQL schema for Formula1 dataset

3. Description of how to replicate the project: creation of the databases, and running the queries

Using both databases, it was achieved the goal to see the team with more wins in a race and the year that happen.
For this, both databases were sort by descending order by “wins”. The data also sorted in descending order for field “year”, to see the most recent data.

3.1. select the dataset and the databases schemes

A schema was performed for each database (relational and no relation databases). The SQL and NoSQL differ in whether they are relational (SQL) or non-relational (NoSQL), whether their schemas are predefined or dynamic, how they scale, the type of data they include and whether they are more fit for multi-row transactions or unstructured data.

For the RDMS was defined the primary Keys: for Constructors Table, the ConstructorId is the primaryKey and the secondary key for Constructor_standings Table. For Table races, raceID is the primary key and the secondary key for constructors_standings.
In both Figures 1 and 2, it can be observed the data type for each field.

3.2. Creation of the databases

For RDBMS, the database was created using sqlite3, where three tables were created for constructors, standings and races and the respective of CSV file were added to Table. To add data from csv files, the data was convert to dictionary and then added to the respective Table.

For noSQL, the database was created in MongoDB, using pymongo.
A database called formula1 was created and three collections were created: constructors, constructor_standings and races. By using pandas, the csv files were read and convert to dictionary to include their data into to the respective collection.

3.3 Creation of queries for each database (relational and noSQL)

For RDBMS:

a) Two simples queries, selecting data from one or two columns/fields

To verify if the csv file data was included in the respective table, the field of interest were selected for each table:

- For constructors: constructorId, name and nationality
- For standings: constructorId and wins
- For races: raceId and year

b) Complex queries, using joins and aggregates, involving at least 2 tables/collections of your database

One query was performed as follows:

```
SELECT standings.wins AS wins, constructors.name, constructors.nationality AS nationality, races.raceId AS raceID, races.year AS year
FROM constructors INNER JOIN standings ON constructors.constructorId =        "standings.constructorId
INNER JOIN races ON standings.raceId = races.raceId
GROUP BY wins ORDER BY CAST(wins "AS INTEGER) DESC, CAST(year AS INTEGER) DESC "
```

Note: The CAST operator is used to convert a value from a data type to another data type, where, for example: value 9 in “wins” were converted to “09”, and, thus, is lower than 19. Without CAST, it was considering value 9 higher than 19, in descending order.

The second one was created as above described, but with indices created for each table.

c) Two update/insert queries

To add data from csv files, the data was convert to dictionary and then added to the respective For these three queries were made: one for each collection.

An example of query for insert for races table:

"INSERT INTO constructors ('constructorId', 'constructorRef', 'name', 'nationality', 'url') VALUES (?, ?, ?, ?,?) ;"

For noSQL:

a) Two simples queries, selecting data from one or two columns/fields

To verify if the csv file data was included in the respective collection, the field of interest were selected for each collection, convert to dataframe and selected the first 5 rows.

b) Two complex queries, using joins and aggregates, involving at least 2 tables/collections of your database

One query was created and included $lookup, $unwind, $sort, $limit and $project, where:

$lookup: The $lookup stage adds a new array field to each input document. The new array field contains the matching documents from the "joined" collection. The $lookup stage passes these reshaped documents to the next stage.

$unwind: treats the sizes field as a single element array if: the field is present, the value is not null, and. the value is not an empty array.

$project: passes along the documents with only the specified fields to present

The second one was created as above described, but with indices created for each table.

c) Two update/insert queries

Data from csv files was converted o dictionary and added to the respective collection. Moreover, for optimization, it was removed non relevant data from each collection, such as, the fields round, circuitsId, lat, date, time, … from collection races. the fields round, circuitsId, lat, date, time,tec… from collection races.

3.4. Indexing and Optimization

a) Implement optimizations and adequate indexing in your databases

For RDBMS:

It was created the following indexes:

- constructorId dor constructors2 table
- constructorStandingId for stanings2 table
- raceId for races2 table
-

For no SQL:

It was created the following indexes:

- constructorId dor constructors2 collection
- constructorStandingId for stanings2 collection
- raceId for races2 collection

The indexes for tables and the addition of $sort and $limit were used for optimization, for a faster achievement of the data.

b) Test the performance of your queries in your databases with prior optimization vs after optimization

For RDBMS:

The test was performed to the queries that implicates the join of the three tables, by create a function “performance”. This function measures the time that the query is executed.

This test was performed before and after optimization and it was obtained 0.01087 and 0.00997, respectively. Following these results, it is demonstrated that the best performance is using indexes, as it took less time to run the query.

For noSQL:

It was tested the performance for the query that evolved to find a constructor team that has the name started with letter A and ends with letter C, before and after optimization. The time spent was de same in both situations. The noSQL can work with structured and non-structured data. The expectation was that with index, it took less time. This can be explained, since in noSQL, indexes can be less robust and less efficient than indexes in SQL databases. As the data is indexed in noSQL database, the purpose is that index data somehow related to one another, like data from the same column. Otherwise, we won’t be able to store it efficiently and the query time will be pretty much the same as without the index. For instance, if you index column A but retrieve data from column B, the index won’t help. For detailed evaluation it can be apply. *explain("executionStats")* to the queries, for more detailed information.

Note: to replicate the databases and re-run the queries again, during the project development, for RDBMS database, it was added “IF NOT EXISTS” for creation of tables and indexes and, for noSQL, a *db.collectionname.drop()* was applied at the beginning of the scrip, before the collection creations.

4. Discussing points done/not done in the project

For SQL, the attribution of index can be a game-changing, for a large amount of structure data. It can be time consume since a schema must be design for relational databases, but, when an index is associated to specific columns, is faster to search for a specific field. SQL databases require more administration than no SQL databases, such as creating and maintaining indexes and Views.

For noSQL, there is the possibility to include various features and is more flexible to work with the data stored. No schema is mandatory and predefined. For example, in our data base, we can consider the field “name” as a key value, which will include the data from “races” and the data from “constructors”. This type of scheme was not performed but was considered.

Data in SQL databases is typically normalized, so queries for a single object or entity require you to join data from multiple tables. As your tables grow, the joins can become expensive. However, data in NoSQL databases is typically stored in a way that is optimized for queries. Queries In noSQL do not require joins, therefore, are very fast.

The dataset for Formula1 also contains data regarding circuits, results, drivers,… o add this information to the database, it will be more easer with noSQL, as no pre-schema needs to be performed. While using SQL database, a schema must be redefined, as well as the indexes, and the tables need to be joined in order to find for a specific object such as the name of driver from Germany who run in Circuit de Barcelona-Catalunya, per example.

Annexes:

Annex A – F1_rdms_no_index

Annex B – F1_rdms_index

Annex C – F1_nosql_without_index

Annex D – F1_nosql_with_index