Graph and Modern Databases

 az5645a@gre.ac.uk | abdullahbinzubair570@gmail.com

Abdullah Bin Zubair

2021

Table of Contents

[Implementation 1](#_Toc70971173)

[**Assignment 1:** Create large JSON document with your own data and corresponding JSON schema. Make sure your JSON document contains various data types’ elements, **demonstrates am appropriate complexity and is valid against your JSON schema.** 1](#_Toc70971174)

[Please provide an appropriate explanation/discussion of your implementation 1](#_Toc70971175)

[**Assignment 2:** Create one database in Redis with your own data (**at least** 10 key/value pairs with values varying in data type) andshow at least 5 different queries on that data. **Make sure that your data and your queries demonstrate an appropriate complexity.** 4](#_Toc70971176)

[Please provide an appropriate explanation/discussion of your implementation 4](#_Toc70971177)

[**Data Insertion:** 4](#_Toc70971178)

[**Assignment 3:** Create one database in Cassandra with your own data (a column-family with at least 5-7 columns and 30-40 rows) andshow at least 5 different queries on that data. **Make sure that your data and your queries demonstrate an appropriate complexity.** 8](#_Toc70971179)

[Please provide an appropriate explanation/discussion of your implementation 8](#_Toc70971180)

[**Assignment 4:** Create one MongoDB with your own data (a collection with at least 20-30 documents of a different structure) and show 10 different queries on that data. **Make sure that your data and your queries demonstrate an appropriate complexity.** 12](#_Toc70971181)

[Please provide an appropriate explanation/discussion of your implementation 12](#_Toc70971182)

[**Assignment 5:** Create one Graph database using either Neo4J or AllegroGraph with your own data (at least 25 nodes with relationships) and show 10 different queries on that data. **Make sure that your data and your queries demonstrate an appropriate complexity.** 17](#_Toc70971183)

[Please provide an appropriate explanation/discussion of your implementation 17](#_Toc70971184)

# Implementation

## **Assignment 1:** Create large JSON document with your own data and corresponding JSON schema. Make sure your JSON document contains various data types’ elements, **demonstrates am appropriate complexity and is valid against your JSON schema.**

## Please provide an appropriate explanation/discussion of your implementation

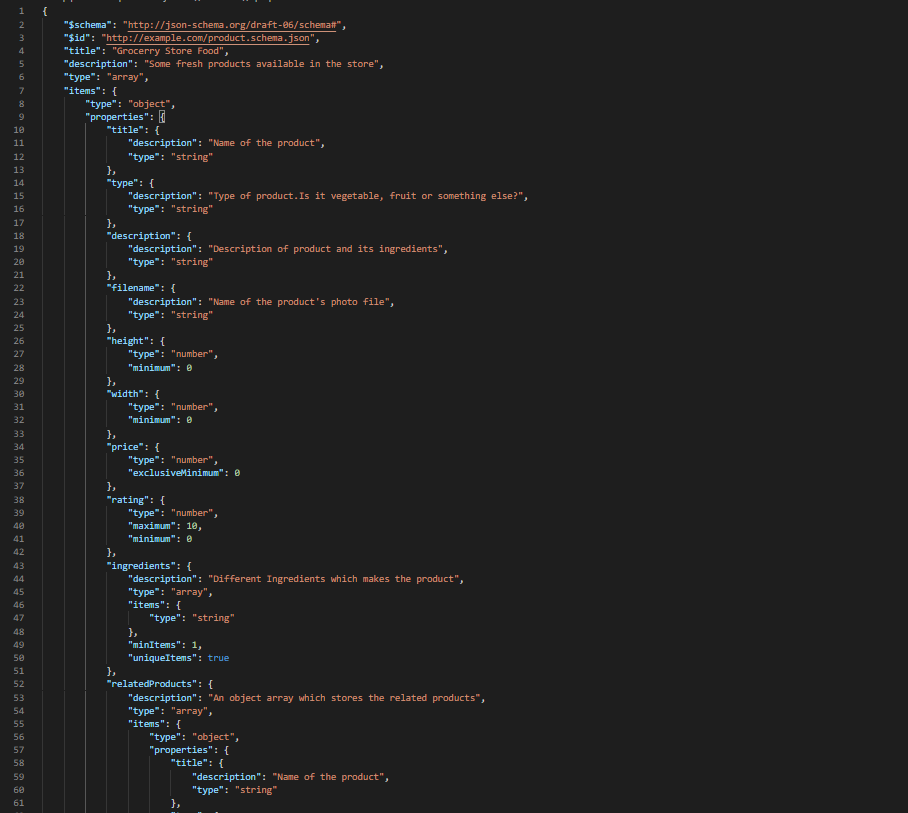
In this implementation I will be constructing a JSON schema and then providing the dataset which will adhere with the constructed schema.

**Explanation of Schema:**

* The schema accepts an **array** of **JSON objects** which are **Grocerry Food Items** and their **details**
* Each object consists of **variable number of key/value pairs but 8 of them are common** in every object
* The required/common keys are**: "title", "type", "description", "filename", "height", "width", "price" and "rating"**
* The optional keys are **"ingredients" and "relatedProducts"**
* "title", "type", "description" and "filename" are **string** type
* "height", "width", "price" and "rating" are **number** type (integer and float)
* "Ingredients" is an **array of string type objects**
* "relatedProducts" is an **array of JSON objects** itself

**Note:** Ingredients array used is common in every food product for sake of simplicity. Related product array is also common in every object for simplicity.

Below you can see screenshot of the schema but the full schema code is also submitted in zip file:



You can also see the validity screenshot from an online JSON validator tool:

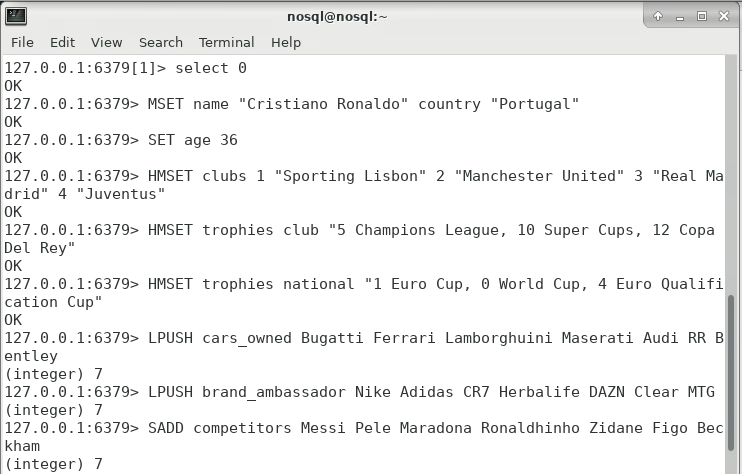


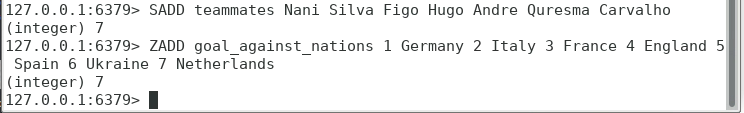
In the submitted zip file, schema file is **schema.json** and data file is **dataset.json.**

## **Assignment 2:** Create one database in Redis with your own data (**at least** 10 key/value pairs with values varying in data type) andshow at least 5 different queries on that data. **Make sure that your data and your queries demonstrate an appropriate complexity.**

## Please provide an appropriate explanation/discussion of your implementation

## **Data Insertion:**





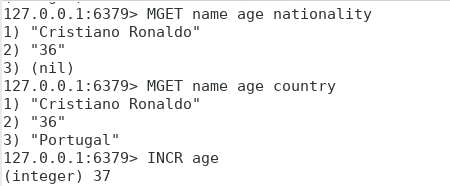
**Explanation:**

The data is about a famous football player, Cristiano Ronaldo. The data is accurate in terms of quality but it’s not complete i.e. the hash with key “trophies” does not include all the Cristiano’s trophies but the ones mentioned are true.

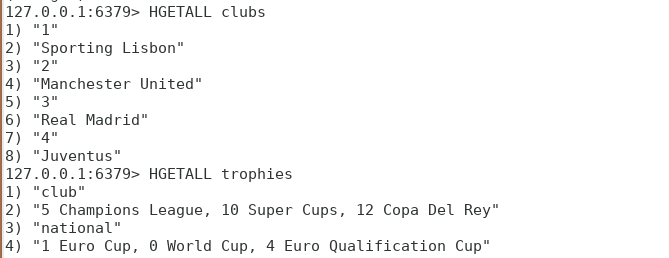
* Key/Value pairs: **name**, **country** and **age**
* Hashes: The **clubs** Cristiano has played for and his **trophies** for **club** and **national** team
* Lists: Cars he has owned (**cars\_owned**) and brands he has been ambassador of (**brand\_ambassador**)
* Sets: Common **competitors** of Cristiano and some of the **teammates**
* Sorted Sets: A set of different nations against whom Cristiano has scored the most goals (**goals\_against\_nations**). The lower the score, more the number of goals against that particular nation
* I have given 7 values for Lists, Hashes’ fields, Sets and Sorted sets because Cristiano Ronaldo always wore No.7 on his shirt.

**Queries:**

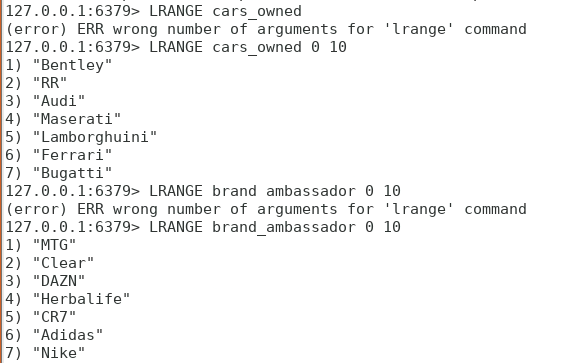
1. Strings and Integers - Following code simply outputs the particular values against the mentioned keys:



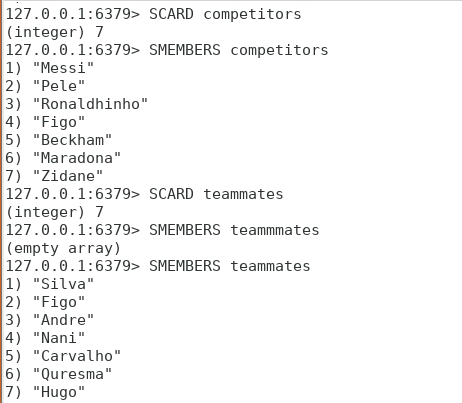
1. Hashes – The HGETALL command gets all the values from the specified Hash Key:

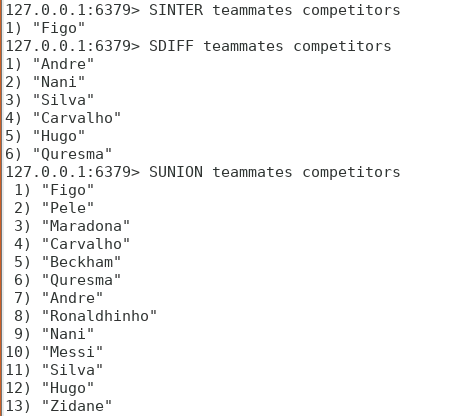


1. List – LRANGE outputs the list values from the given range:

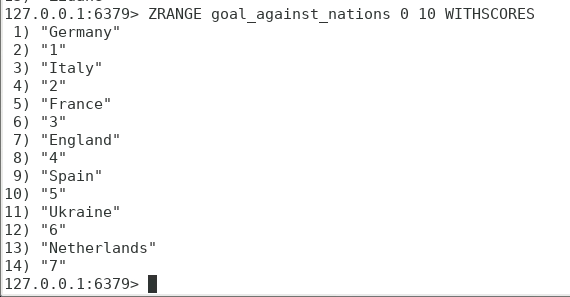


1. Sets – Following you can see use of SCARD for counting members in sets, SMEMBERS for displaying all members and most common set functions (INTER, UNION and DIFF). SINTER displays the intersection of two sets, SDIFF displays the first set’s members subtracted by the second set’s and, SUNION displays all the unique elements in both sets:





1. Sorted Sets – ZRANGE acts same as SRANGE except it can display the score if you insert the keyword ‘WITHSCORES’ in the end of command:



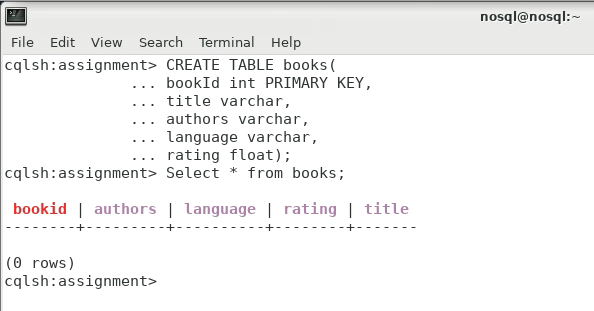
**Extra point:**

* Redis also provides the ability to expire a value after its expiry time. For example in case of shopping cart, it can expire after some set time i.e. 15 minutes if it is left unattended. One of the implementation could be like: SETEX key value ExpiryTime (seconds). The value does not exist in database after its expiry time.

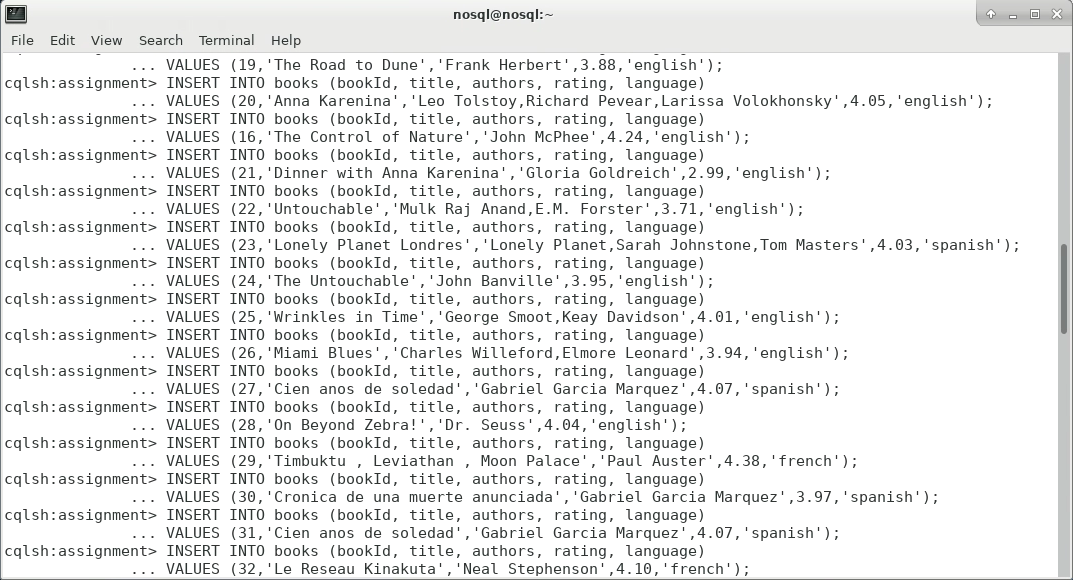
## **Assignment 3:** Create one database in Cassandra with your own data (a column-family with at least 5-7 columns and 30-40 rows) andshow at least 5 different queries on that data. **Make sure that your data and your queries demonstrate an appropriate complexity.**

## Please provide an appropriate explanation/discussion of your implementation

**Table Creation:**



**Data Insertion:**

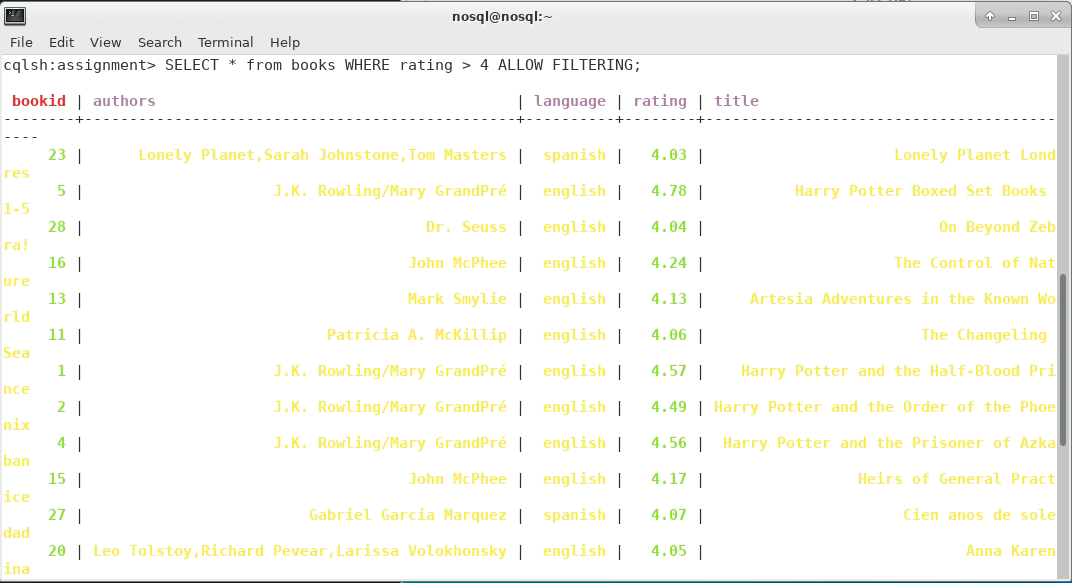


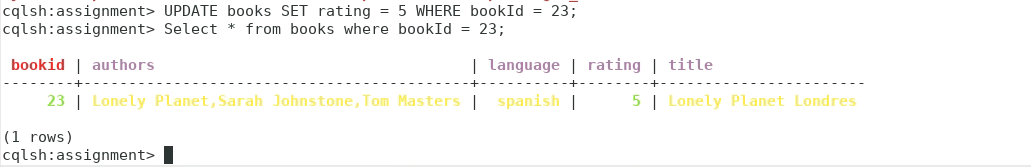
**Explanation:**

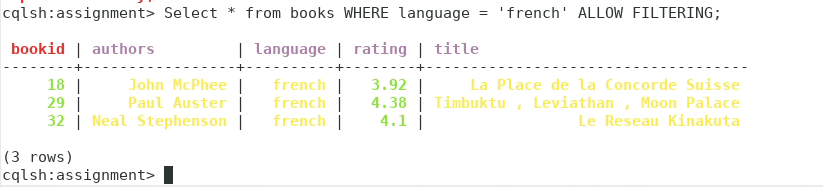
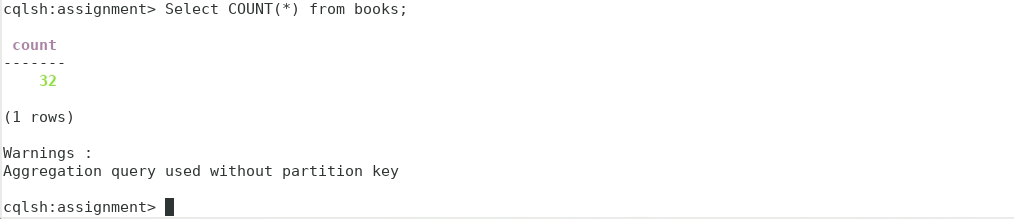
This data is about some of the major books released in recent years. It includes the title, author/authors, rating and language of each book. Rating is floating point number while every other column type is varchar. BookId is primary key and hence it is integer type. This dataset has been completed with the help of an online source: <https://www.kaggle.com/jealousleopard/goodreadsbooks>

**Note**: The above script is included in zip file.

**Queries:**

1. SELECT \* FROM books; - This query is self explanatory as it lists down all the records from the specified table
2. Select \* from books WHERE rating > 4; - This query uses the WHERE keyword to get a set of records which satisfy the given case
3. UPDATE books SET rating = 5 WHERE bookId = 23; - This query is used to update specific column value of a specific row/record

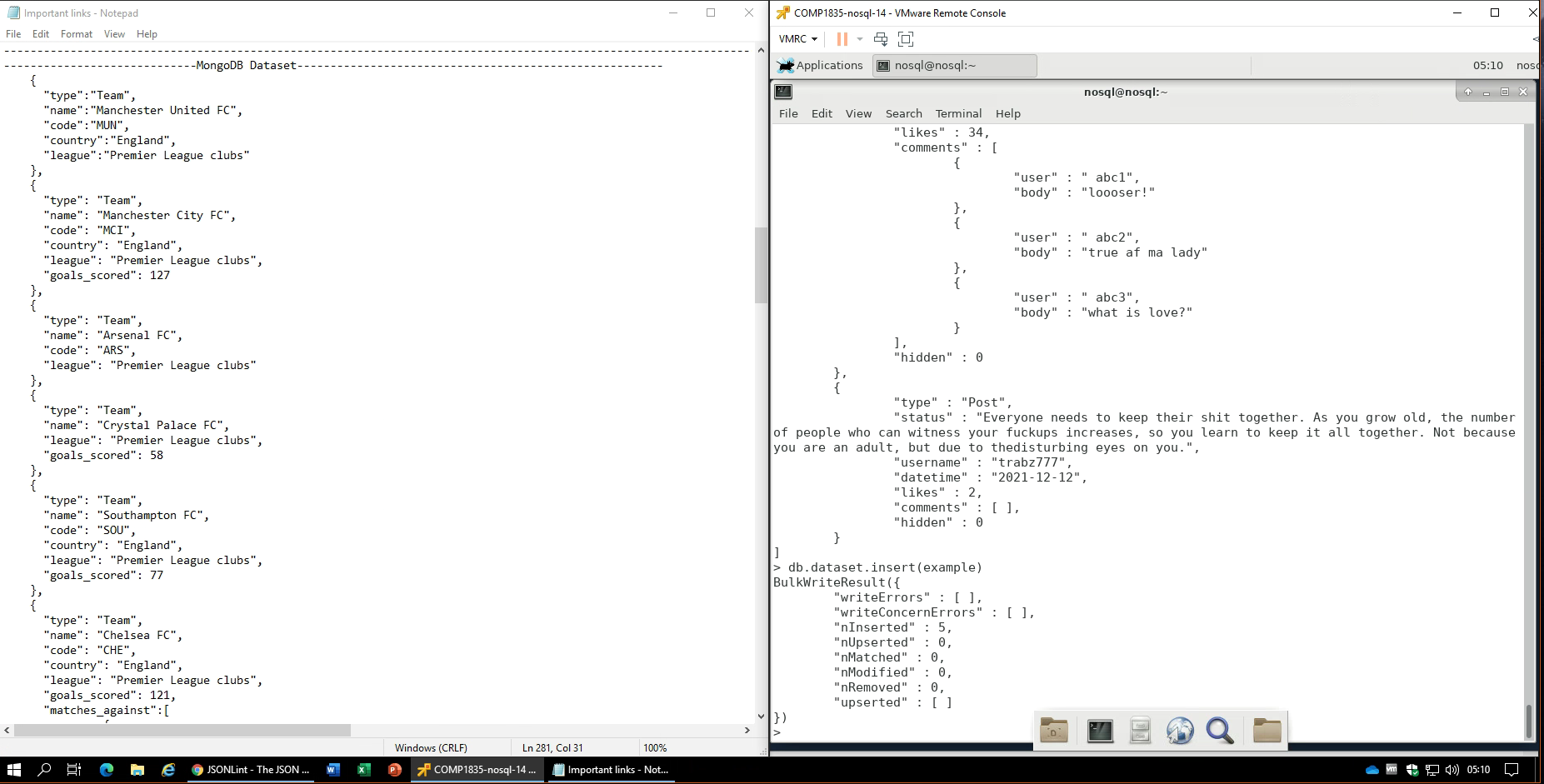


1. Select \* from books WHERE language = ‘french’; - This query also takes advantage of WHERE key word to satisfy a case
2. Select COUNT(\*) from books; - This is aggregation function used to count all the entries in table
3. DELETE FROM books WHERE bookId = 32; - This query is used to delete a certain row from the table

## **Assignment 4:** Create one MongoDB with your own data (a collection with at least 20-30 documents of a different structure) and show 10 different queries on that data. **Make sure that your data and your queries demonstrate an appropriate complexity.**

## Please provide an appropriate explanation/discussion of your implementation

**Data Insertion:**



**Explanation:**

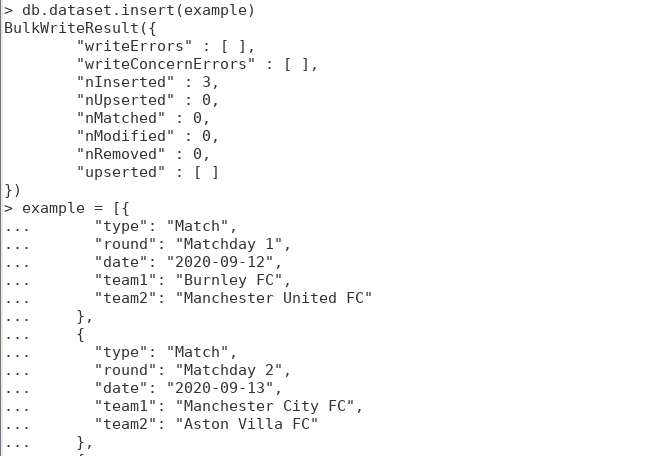
This is an example of un-structured data as it contains three type of documents**, Teams, Matches and Posts** with different number of fields/columns/keys and values

* **Teams**: England football club teams’ data (6)
* **Matches**: Some of the matches’ data which were played in first few weeks of league (9)
* **Posts**: Random Facebook posts with comments (5)
* All the documents can have different amount of fields, data types and structure.
* Data types include **Numbers (integers), Strings, Arrays and Boolean**.

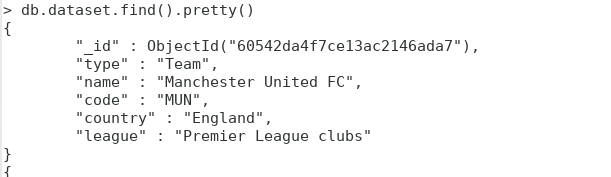
**Note**: Dataset is included in the submitted zip file.

**Queries:**

1. Create collection/Insert documents – When you insert a document into a new collection which does not exist before, then the following command creates the collection and inserts the documents as well. The ‘example’ variable stores an array of documents.



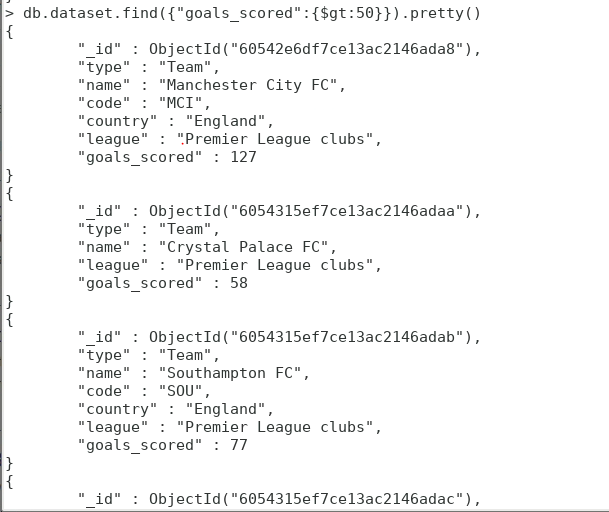
1. Selecting all the documents in a collection



1. Select documents on basis of a criteria – for example selecting all the matches from dataset



1. Select documents on basis of a criteria – for example selecting all teams who have scored more than 50 goals



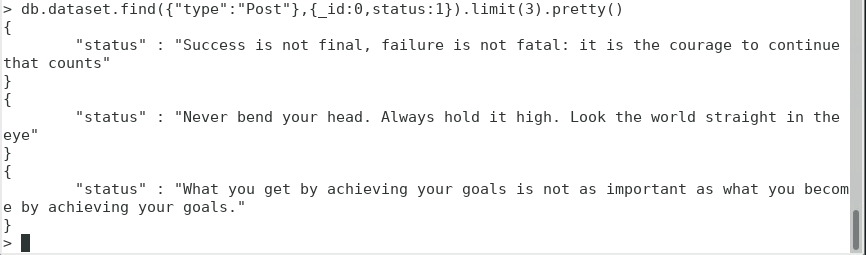
1. Select documents on basis of criteria – for example selecting all posts with more than 10 likes



1. Updating a value in database



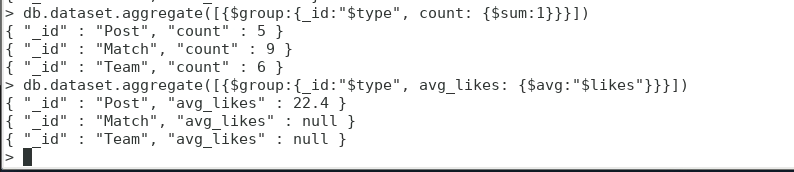
1. Using Projection and limiting functions of Mongo Shell – finding first three posts and displaying only their status



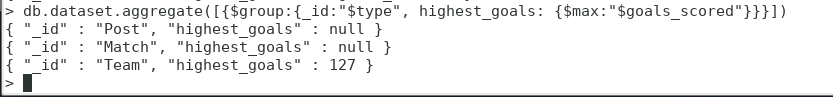
1. Displaying all teams sorted data on basis of goals scored, ascending order. Note that first 2 documents do not have the goals data so that value for them is 0 that’s why they are on top of result set



1. Aggregating – Counting (sum) total number of different type documents and calculating (avg) average number of likes on the posts. Note that all matches and teams do not have field of likes so in the second query the summation for them is null



1. Aggregating – Select maximum number of goals scored by each team. Note that as posts and matches do not have a ‘goals\_scored’ field, their maximum generated is null.



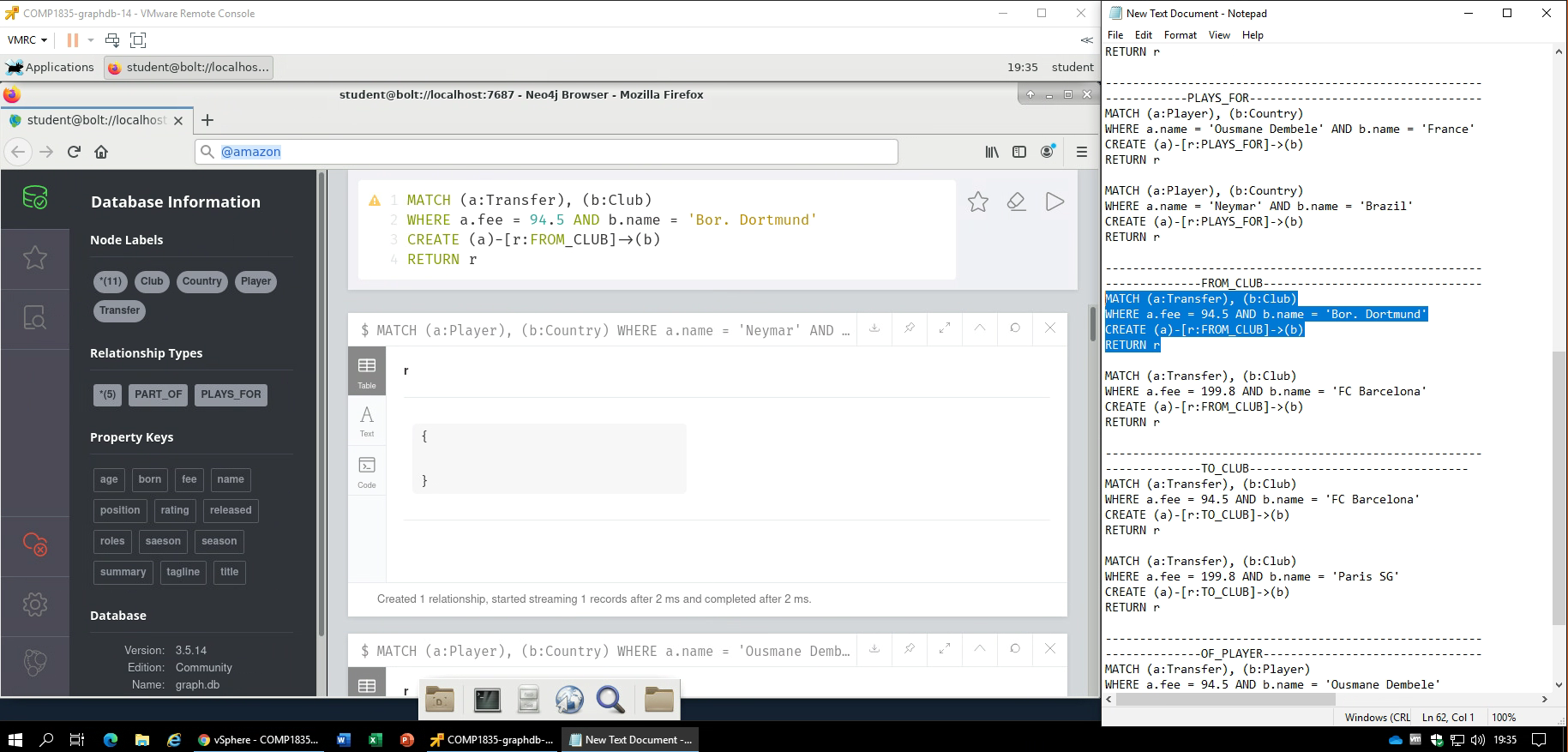
## **Assignment 5:** Create one Graph database using either Neo4J or AllegroGraph with your own data (at least 25 nodes with relationships) and show 10 different queries on that data. **Make sure that your data and your queries demonstrate an appropriate complexity.**

## Please provide an appropriate explanation/discussion of your implementation

**Plan:**

For this assignment I will be using Neo4j as it is an exceptional implementation of LPG (Labelled Property Graph) database. In the beginning I will create a dataset containing relatively small amount of nodes and then run the queries on it. After that I will increase the dataset to the required limit and then run some important queries again.

**Data Insertion:**



The inspiration for this dataset creation has been taken from Neo4j’s own documentation for developers: <https://neo4j.com/graphgist/cd8868d1-da9a-44ad-a221-baab3086c902#_analysing_football_transfers_with_neo4j>

Only the idea has been taken. All the CRUD operations and queries has been started from scratch.

**Explanation:**

This dataset represents the football player transfers and the relationship between different entities involved.

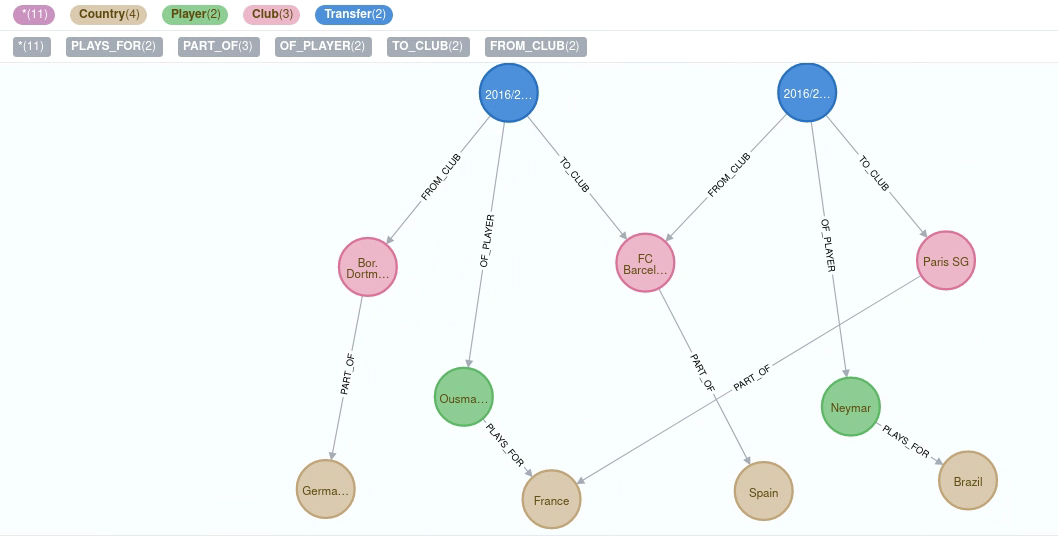
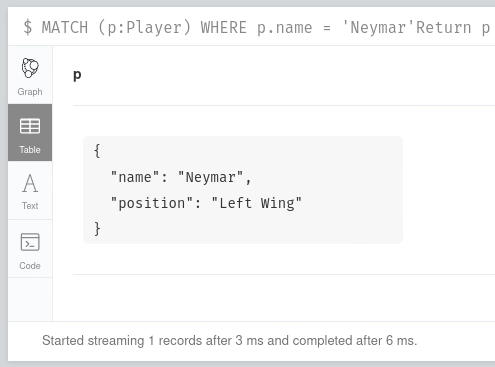
**-> Node Labels**: Player, Club, Transfer and Country

* **Player**: Nodes represent the transferred players and their data. In this implementation there is only one property other than ‘name’ which is the ‘position’ where they play on the field.
* **Club**: Nodes represent the clubs associated with the transfers.
* **Transfer**: These nodes represent the actual transfers’ data. In this case it is the transfer ‘season’ and ‘fee’.
* **Country**: These are the countries of players and the clubs from where they belong.

**->Relationships:** All nodes are connected with one or more than one node with relationships. In this implementation there are 5 types of relationships namely: PART\_OF (club’s country), PLAYS\_FOR (player’s country), FROM\_CLUB (player’s present home), TO\_CLUB (Player’s destination), OF\_PLAYER (The player).

**->Constraints:** Community edition available does not support majority of constraints’ functionality that’s why in this implementation there is just UNIQUE constraint asserted on node’ property.

**Queries:**

1. Visualizing the full schema – CALL db.schema.visualization()****
2. Retrieving player by their property name 
3. Listing fees of transfers happening in season 2016/2017 and 2017/2018



1. Listing all the transfers with transfer fee of more than 100 million GBP and happening in season 2016/2017



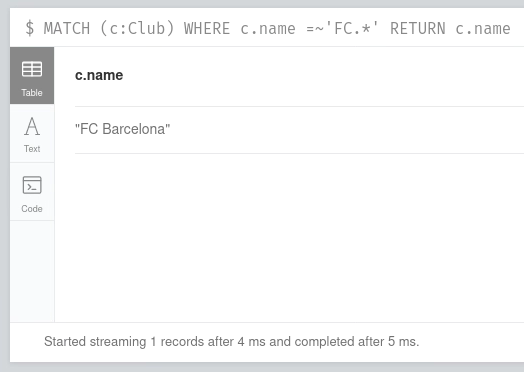
1. Listing out all the countries ordered by their names



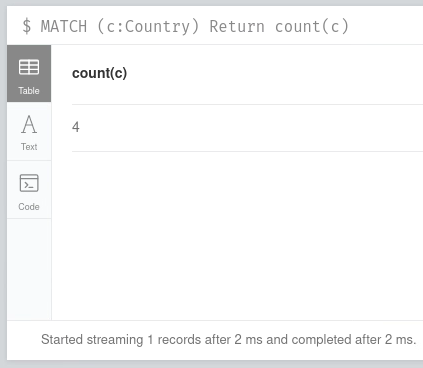
1. Listing all the clubs starting with ‘FC’



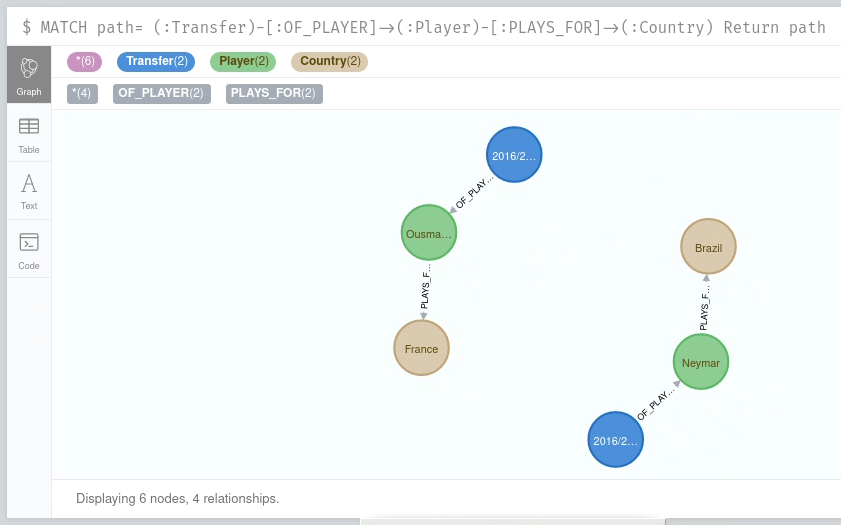
1. Retrieving same result with use of Regular expression



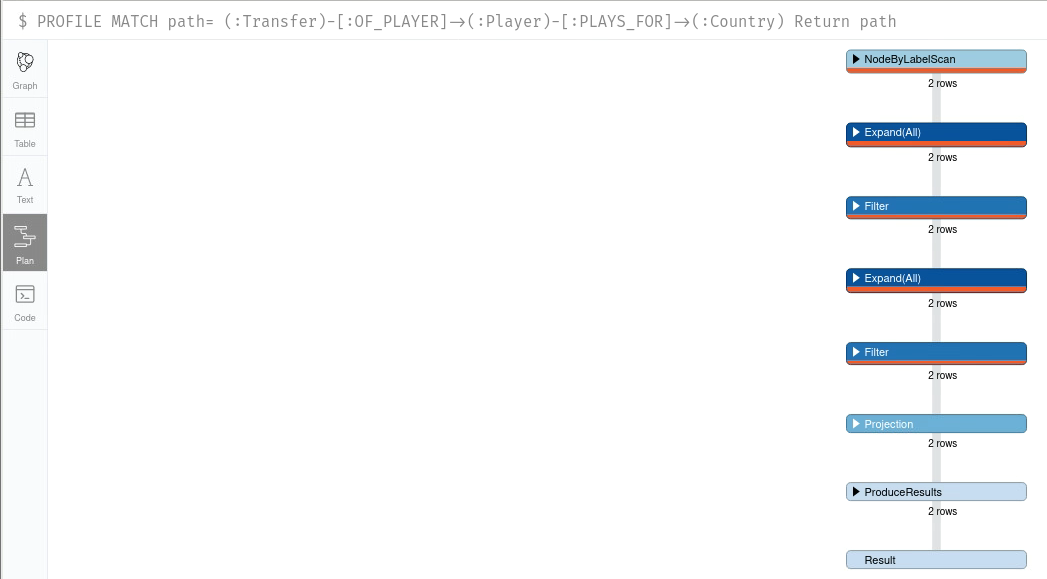
1. Using aggregate function to calculate total number of countries in database



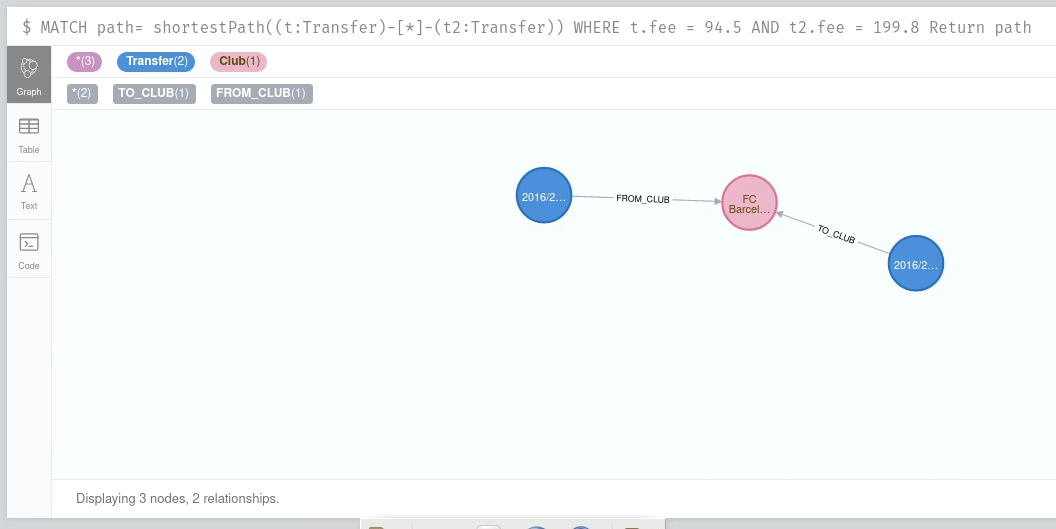
1. Traversing the path of player transfer to which country he plays for



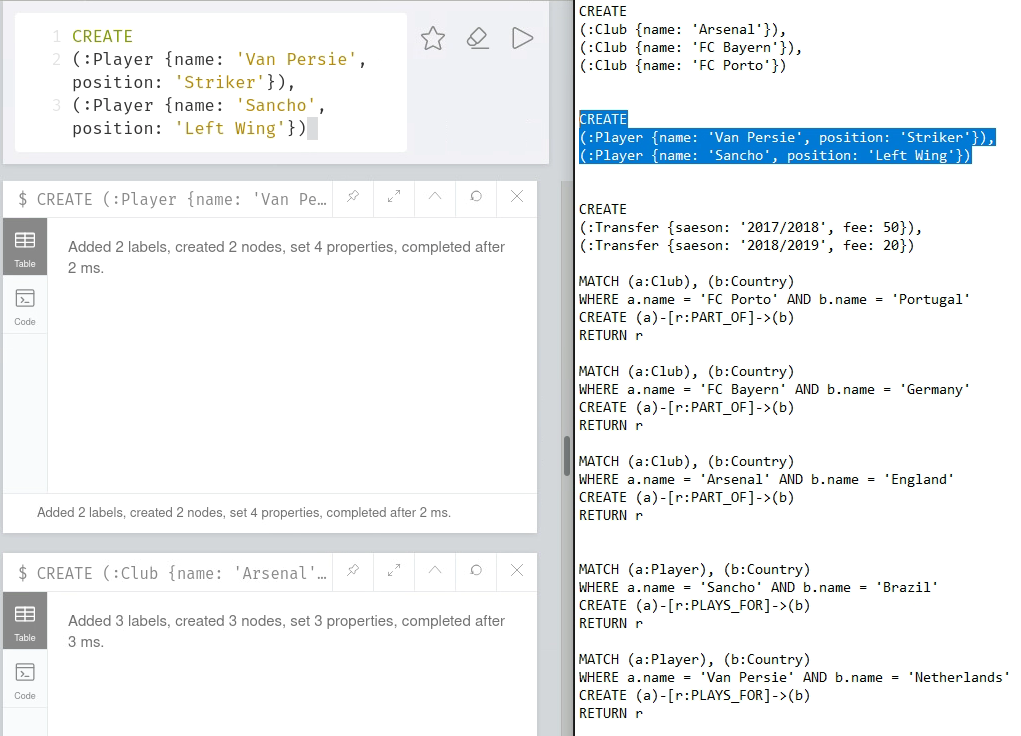
1. Analysing the query using PROFILE keyword



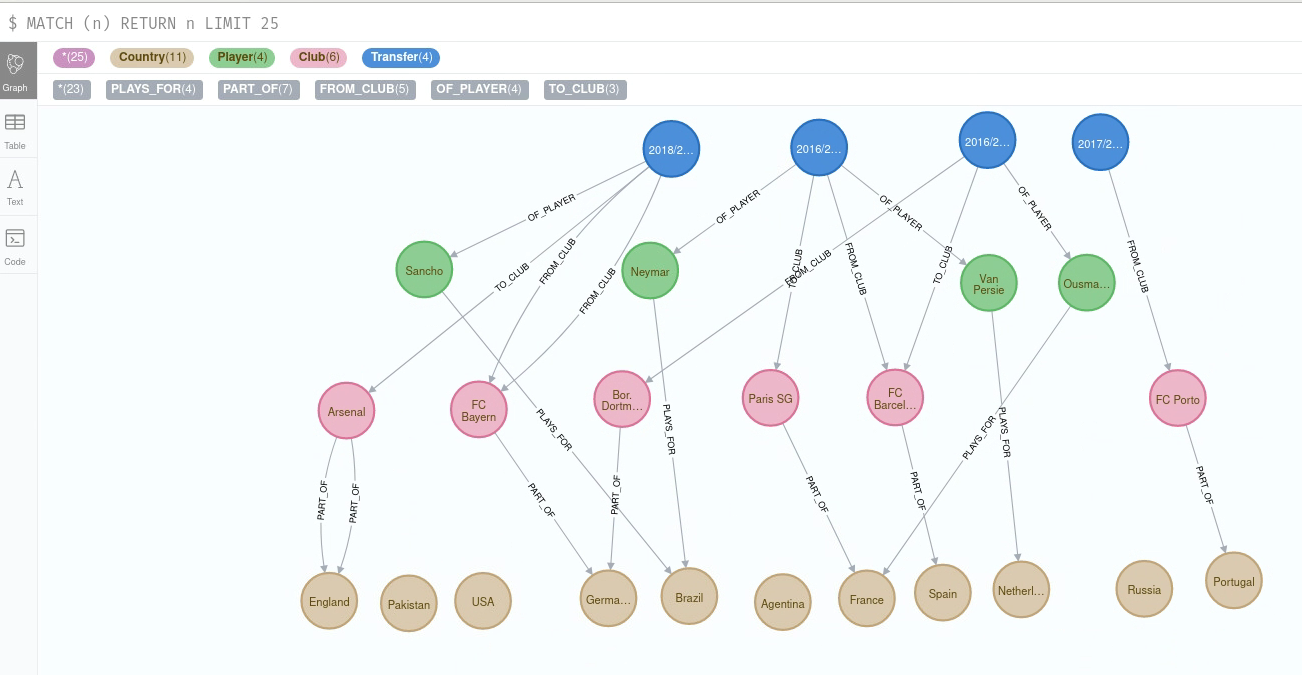
1. Finding the shortest path between 2 transfers



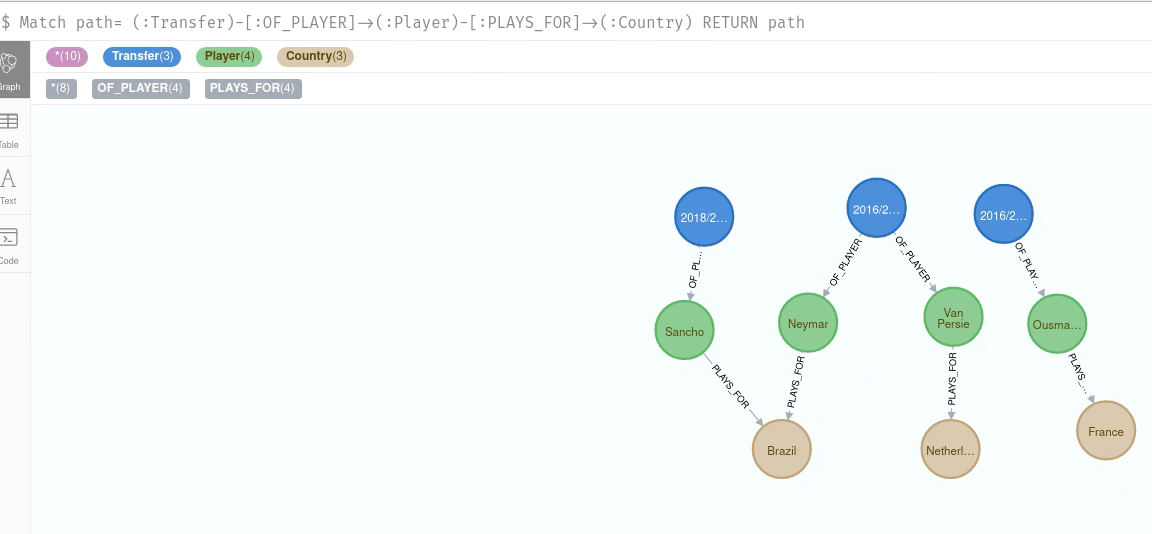
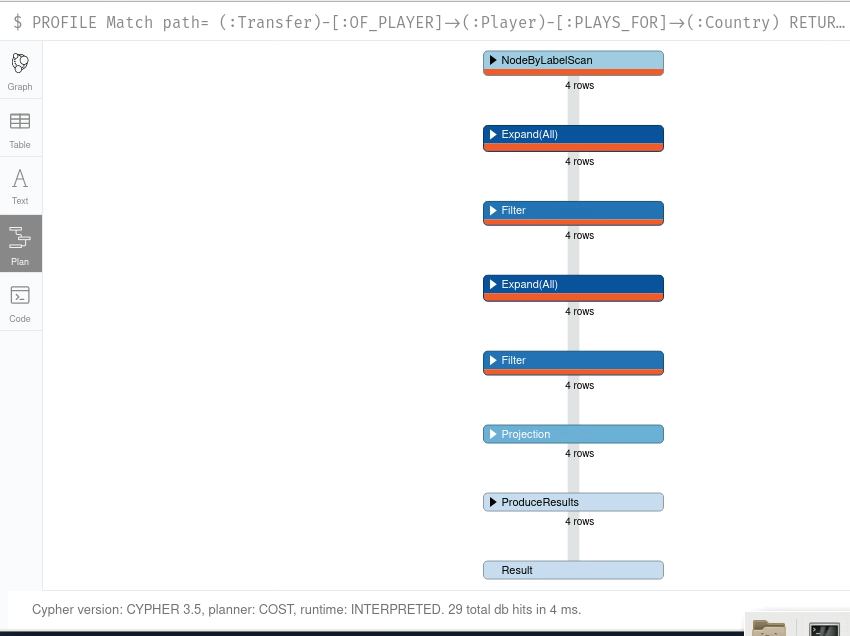
**Increasing the Dataset:**



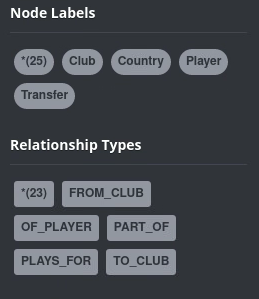
**Schema after addition:**



Now I’m going to run query number 9 and 10 again with the final dataset:

* 9) 
* 10)

**Dataset after addition:**



**Note:** In the submitted **zip file**, the full script for creating this database and then adding more data to it has been included as name of **Neo4j Script.txt** (It’s a set of queries).