2020

CSY3024 Assignment - 2

MONGODB AND NOSQL

SHRIHARI MURALITHARAN - 17451158

1. Table of Contents

2. Introduction

The motivation behind this report is to document the following: learning outcome for each lecture conducted, import the EPL 2018-2019 dataset and query the given commands in MongoDB and an essay on the topic “Big Data Analytics and the role of NoSQL”.
The learning diary contains the exercises completed each week and a reflection on the weeks lecture. The next section documents the thought process behind creating and querying the EPL dataset. This section contains images of the code and output, the code in editable format can be found in the appendix.
The final part of this report contains an essay on “Big Data and the Role of NoSQL”.

3. Learning Diary

This section documents the exercises completed, problems faced and learning outcomes of each week.

3.1. Week 17 – Introduction to MongoDB

The lecture on week 17 was an introduction to MongoDB a type of NoSQL database. The following information was gained from the lecture:

- MongoDB is a Document-based and does not use joins making it significantly faster than SQL.
- MongoDB consists of documents which store data, and this is an example structure of a document: *{name: "Shrihari", age: 22, student_no: 17451158}.*
- A document is essentially a set of key-value pairs, and a set of documents is called a collection.
- Keys in a document are strings which must not contain the null character (/0), and values can be of any type supported by Mongo.
- MongoDB supports replication (copying of data to different servers making data easily available in case of failures) which provides eventual consistency.
- MongoDB provides Auto-Sharding (Sharding is the process of storing data in multiple servers instead of a single server), making MongoDB great for Horizontal Scalability.
- MongoDB, unlike SQL, does not need a predefined Schema and each document can have different kind of data.

Mongo provides APIs for multiple programming languages.

3.1.1. Lab Tasks

1. *Comparison of MongoDB and relational databases.*

MongoDB	Relational Databases
MongoDB uses documents to store data. A document has the following format: {key: "value"}. A group of documents is known as a collection.	Relational databases store data using tables. A row from the table is known as a tuple and a column is known as an attribute. A group of tables is known as a database.
MongoDB does not require a schema, which means the structure of a database can be changed easily when the need arises.	Relational databases need a schema to be made before the database is created. This makes relational databases rigid and making changes once the database is built is harder.
MongoDB also provides consistency.	Relational databases provide data consistency.
Complex and unstructured data can be stored easily.	Storing unstructured data is quite complex.
Joins are not required.	Joins are required to access data from two or more different tables.
Made for horizontal scaling.	Best for vertical scaling.

2. *Comparison of MongoDB and other NoSQL databases.*

MongoDB being a NoSQL database shares some common features with other NoSQL databases. Some of the common features are as follows:

1) NoSQL databases do not require a schema.
2) They are made for horizontal scalability.
3) They can store structured, unstructured and semi-structured data.
4) Most of them provide support for sharding and replication.

The differences between these databases are mainly the way they store data and in the way each of them handles relationships. MongoDB is a document data store; other types of data stores include graph and key-value. Graph databases store values in nodes. These nodes are connected by edges which store the type of relationship between the nodes. Graph databases work great when there are many relationships and also when the relationships are complex. Key-value data stores, as the name suggests, store data in key-value pairs. This data store is known as the simplest kind of NoSQL database.

3. *Doing basic tasks in MongoDB*

One of the lab tasks given in week 17 was to run MongoDB and to do some basic tasks to get used to the database. The following screenshots illustrate create database, insert data, update data and querying of the database. These commands were executed using the mongo shell on a terminal.

Figure 1 - Create Database, insert data, query data and use projection

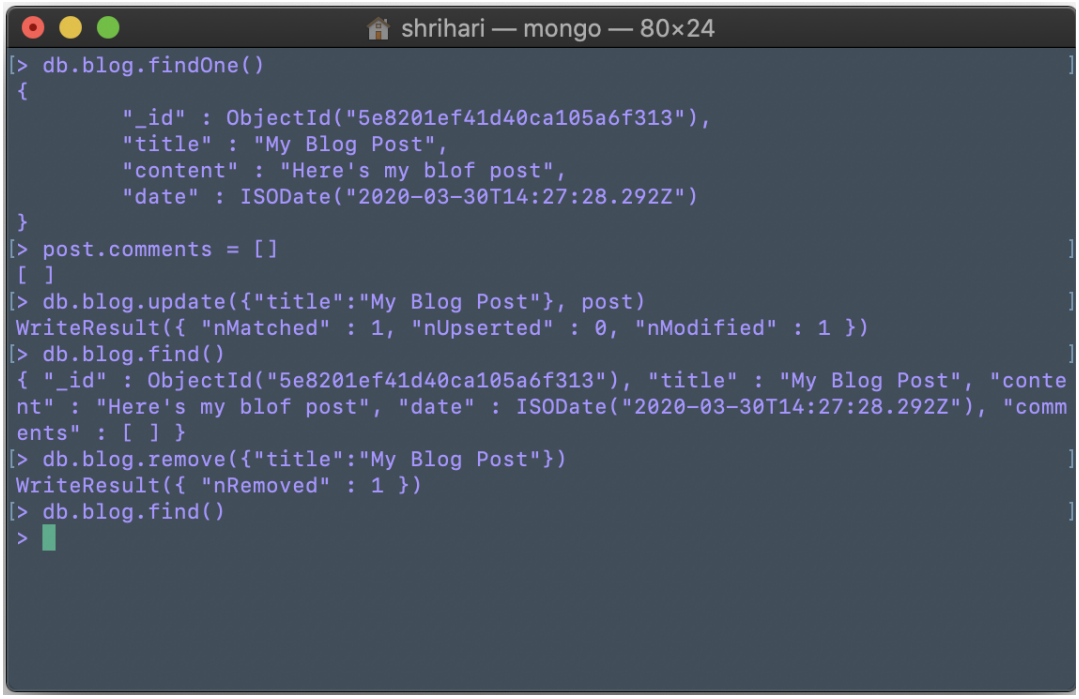


Figure 2 - Find one element, Update and delete

Figure 3 - Insert, Query and Update

3.1.2. Weekly Reflection

The lecture this week helped gain knowledge on MongoDB and its advantages over Relational Databases. Initially, the concept of document and collection was hard to grasp, but these terms were understood once some basic tasks were done in MongoDB.

3.2. Week 18 – MongoDB continued

During week 18 some topics from week 17 were reiterated to help us understand MongoDB better. The new topic covered this week was datatypes in MongoDB, which was quite easy to understand since the datatypes are similar to the ones available in other programming languages.

3.2.1. Lab Tasks

1. Exercises from "Mongodb_tutorial.pdf" (Page 12-33)

"Mongodb_tutorial.pdf" was referred to for additional reading on MongoDB. The book included some exercises such as: create database, create collection, insert, update, query and projection. These exercises were done in the mongo shell on a terminal. The screenshots of the exercises are shown below.

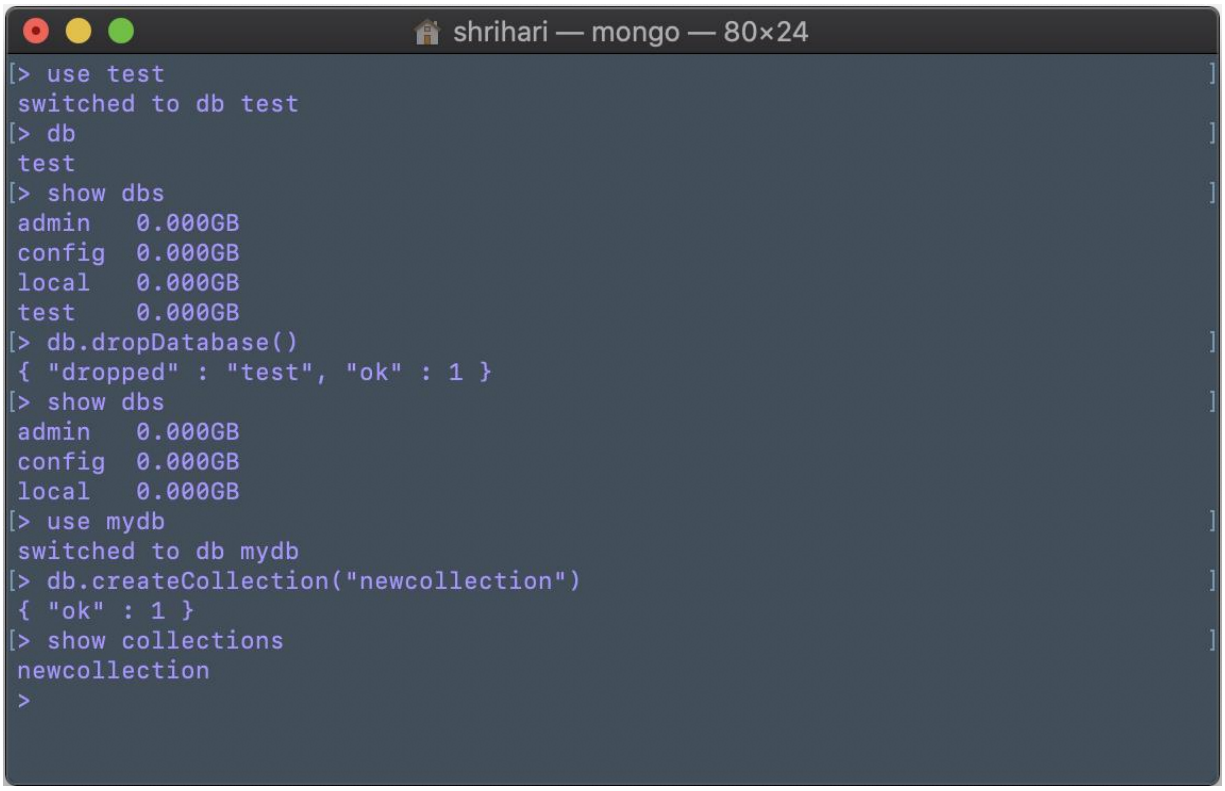


Figure 4 - Create, view and delete Databases, and Create Collection

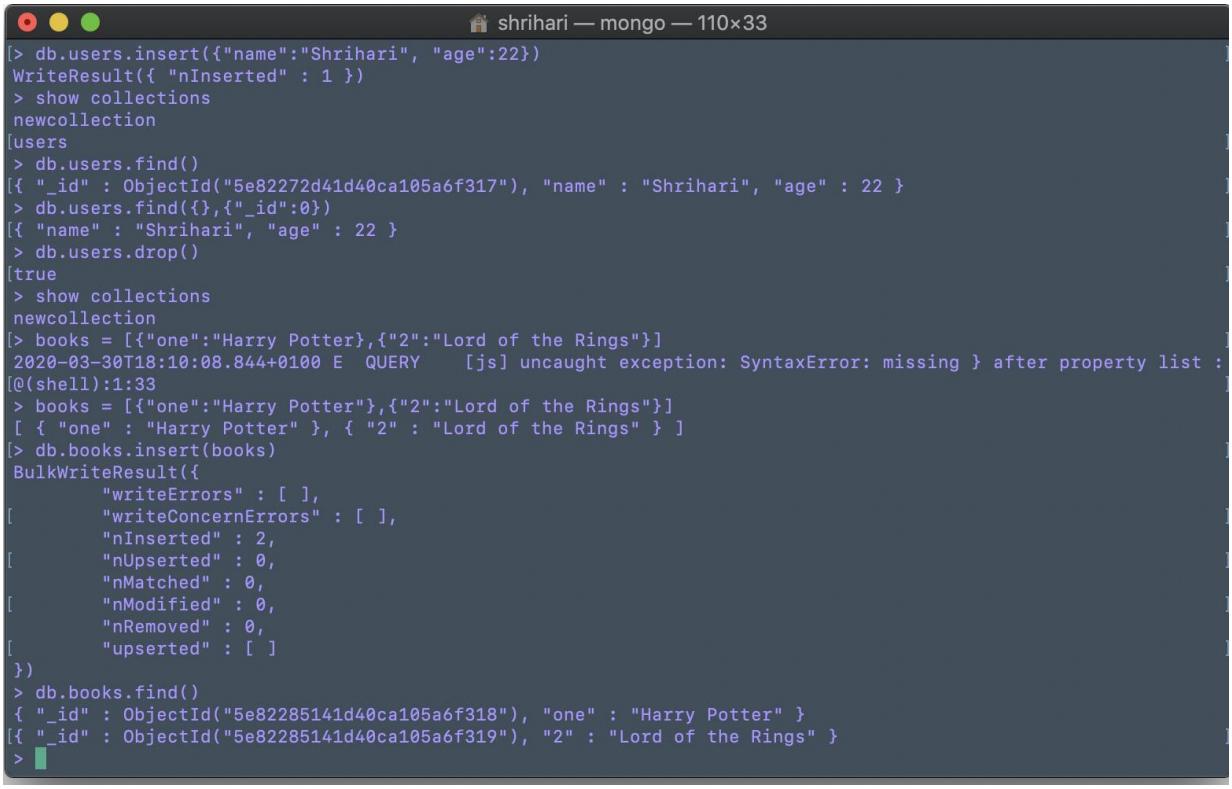


Figure 5 - Insert data and Query Collection

Figure 6 - Insert, Update and Save

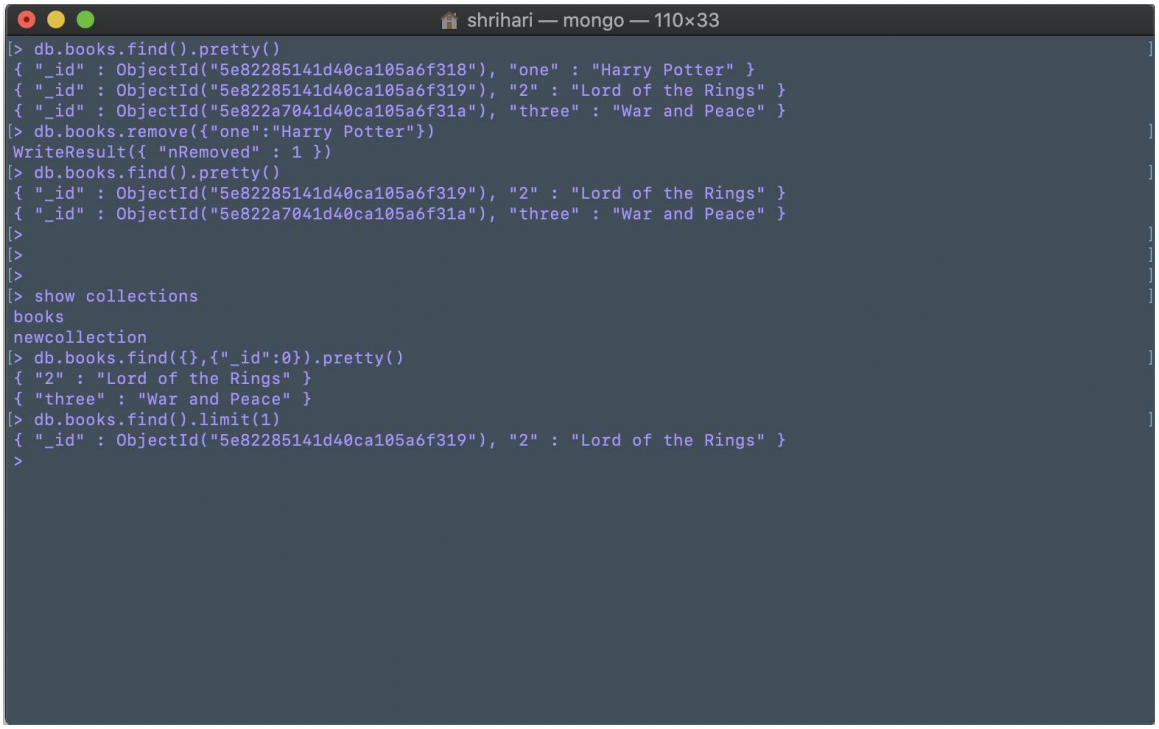


Figure 7 - Query and Projection

2. Import the Restaurants data and query data

1) Write a query to display all documents in the collection restaurants. (Xue, 2020)

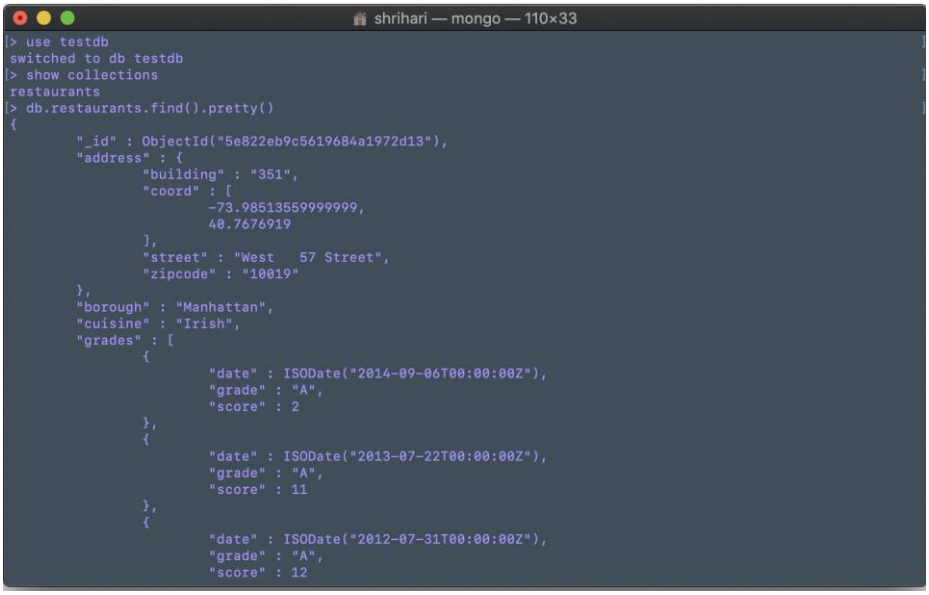


Figure 8 - Display Documents in Restaurant collection

2) Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the documents in the collection restaurant. (Xue, 2020)

Figure 9 - Display selected fields

3) Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine, but exclude the field _id for all the documents in the collection restaurant. (Xue, 2020)

Figure 10 – Exclude ID Field

4) Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant. (Xue, 2020)

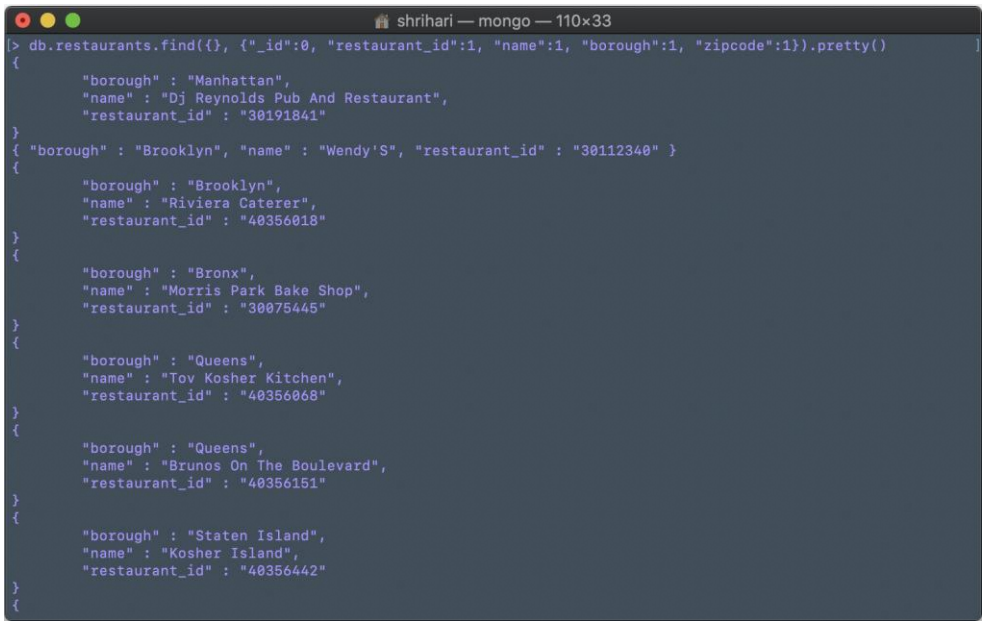


Figure 11 - Display zip code

5) Write a MongoDB query to display all the restaurant which is in the borough Bronx. (Xue, 2020)

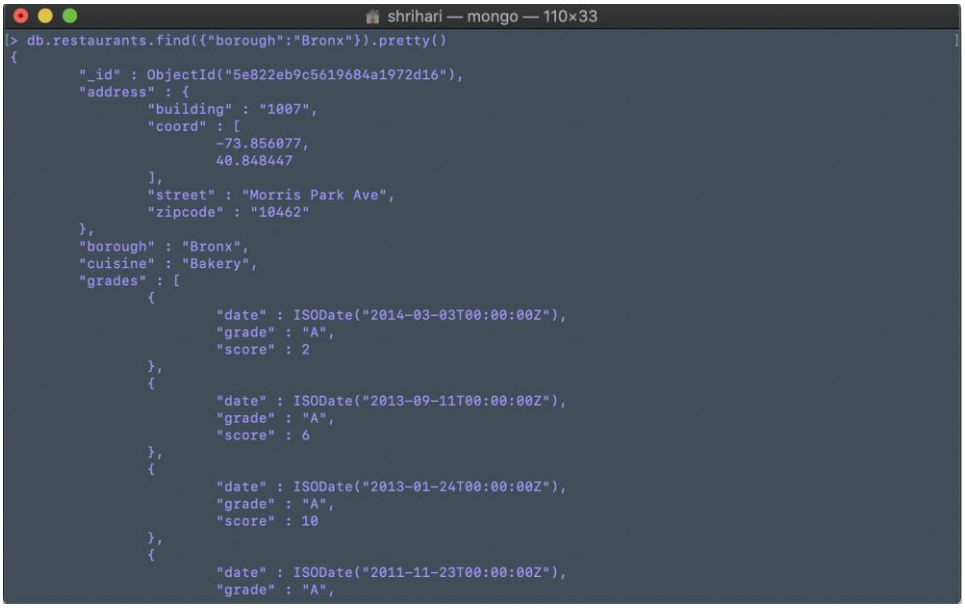


Figure 12 - Restaurant in Bronx

6) Write a MongoDB query to display the first 5 restaurant which is in the borough Bronx.

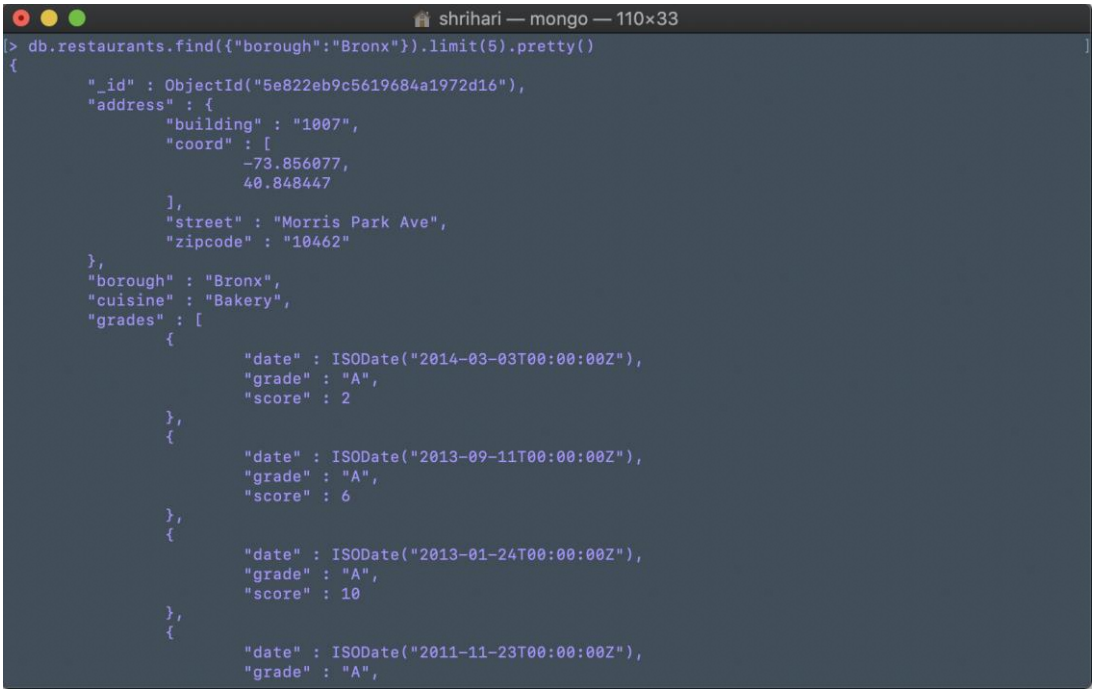


Figure 13 - First 5 Restaurants in Bronx

7) Write a MongoDB query to display the next 5 restaurants after skipping first 5 which are in the borough Bronx.

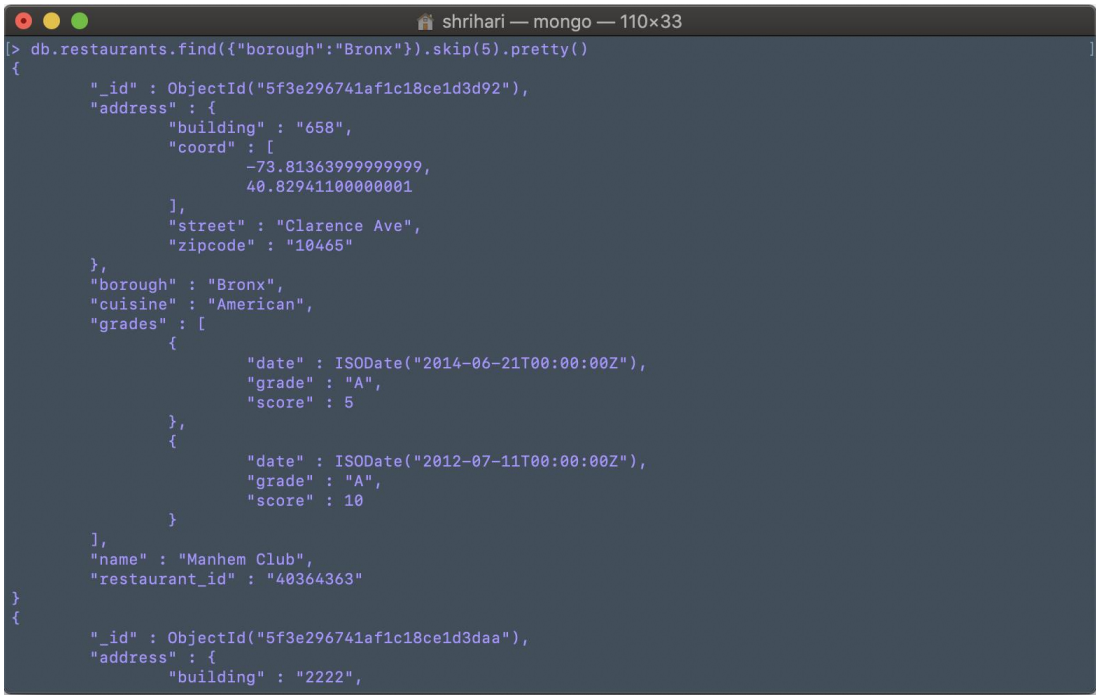


Figure 14 - Skip 5 Restaurants Bronx

8) Write a MongoDB query to find the restaurants that achieved a score, more than 80 but less than 100.

Figure 15 - Restaurants with score>80, but less than 100

9) Write a MongoDB query to find the restaurants which locate in latitude value less than -95.754168.

Figure 16 - Restaurants in latitude less than 95.754168

10) Write a MongoDB query to arrange the name of the restaurants in ascending order along with all the columns.

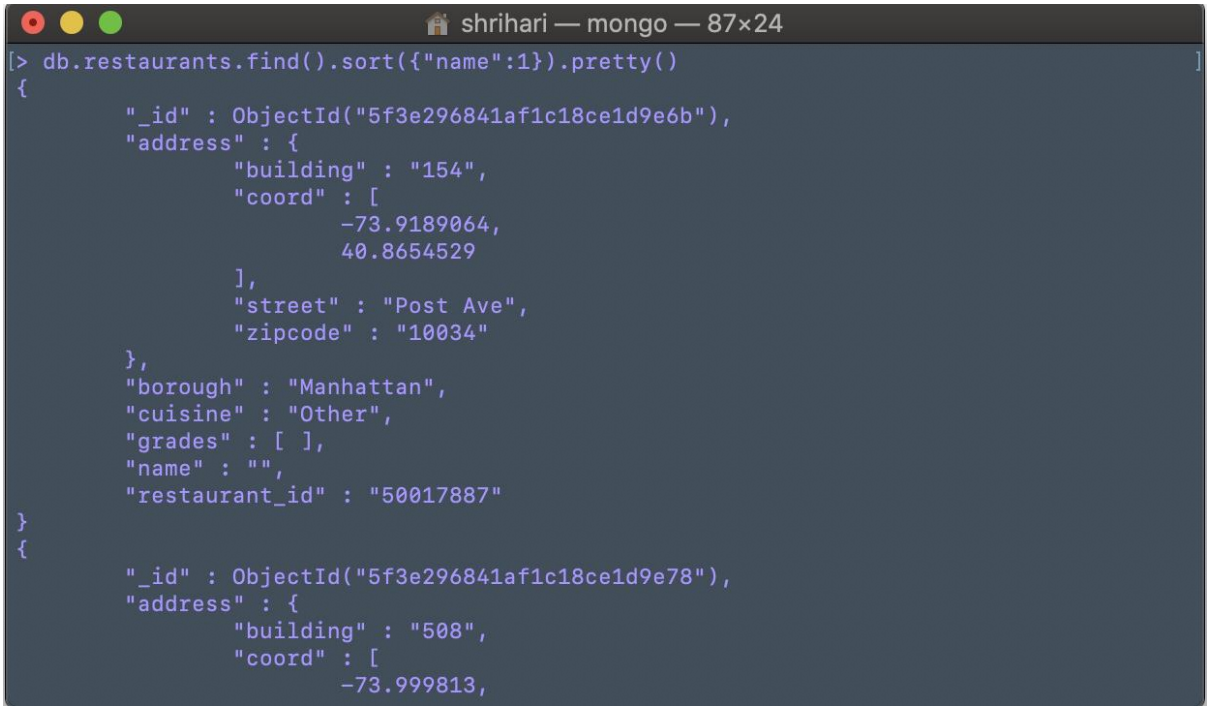


Figure 17 - Restaurants name is ascending order

11) Write a MongoDB query to arrange the name of the restaurants in descending along with all the columns.

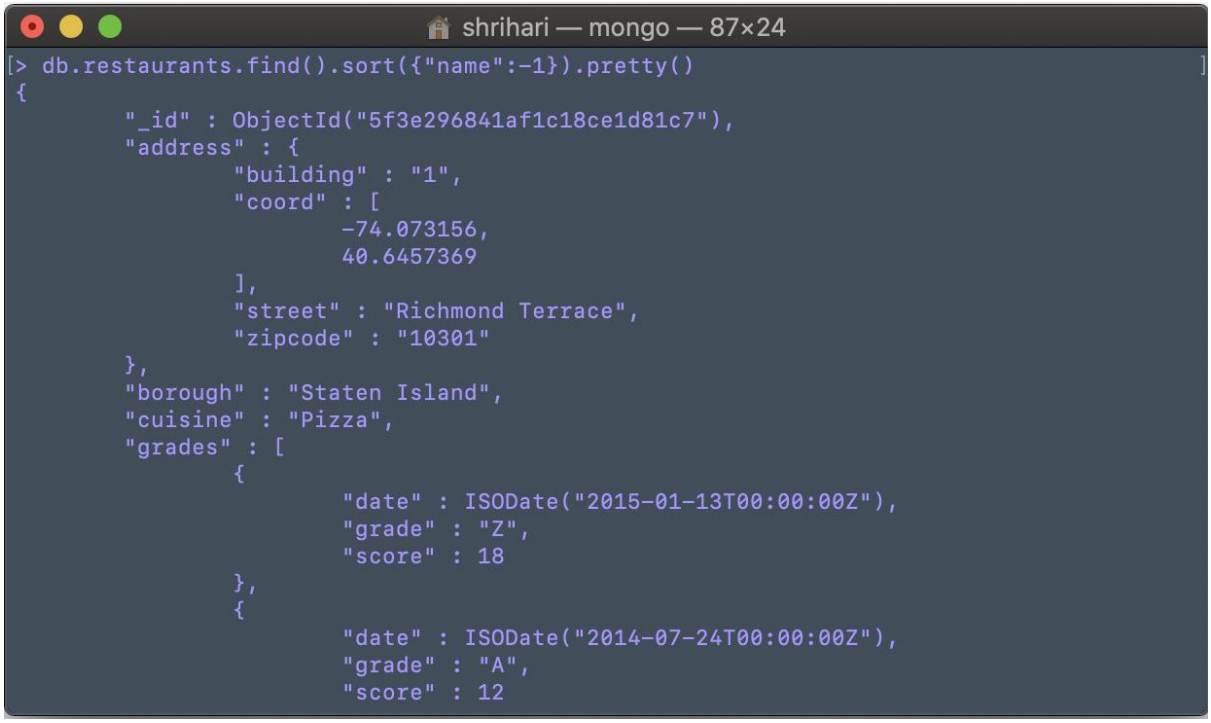


Figure 18 - Restaurant name in descending order

12) Write a MongoDB query to arrange the name of the cuisine in ascending order and for that same cuisine borough should be in descending order.

Figure 19 - Query 12 Week 18

3.2.2. Weekly Reflection

The main focus this week was on practical exercises on MongoDB. These exercises helped the author understand Mongo better. Some concepts such as documents, collections and querying of data were not clear last week, but on completing the exercises these topics were clearly understood.

3.3. Week 19 – MongoDB Modelling

During this week's lecture, the main focus was on modelling approaches for MongoDB. Several models are available for MongoDB and the model can be chosen based on the data that is going to be stored in the database.

There are two main concepts associated with MongoDB data model. This first one is embedded documents, which is storing of documents inside other documents. The more important document is the outer document and the document with lesser importance is the inner document. The following is an example of an embedded document:

Student: {_id: 12345
Name: "Shrihari",
StudentNumber: 17451158,
Address: {
City: "Northampton",
Country: "United Kingdom"
}
}

In the above example, Student is the more important document and it stores address which is an embedded document. This model is used when the size of data is known and if it is also known that the embedded document would not increase by a lot.

Another concept for modelling data is using references. This concept is similar to foreign keys in a Relational database. In this type of modelling one document stores the ID of the other document that it is linked to. This type of model is known as the normalised model, an example can be found below.

Student: {
_id: 12345
Name: "Shrihari",
StudentNumber: 1745118
}
Address: {
Student_id: 12345,
City: "Northampton",
Country: "United Kingdom"
}

This kind of data model is used when it is known that the data is going to increase in volume.

3.3.1. Weekly Reflection

This week's lecture helped the author understand how data can be structured in MongoDB. It was understood that since mongo does not require a schema, the documents can have many structures which allow different kinds of data models. It was also understood that the data models can be selected based on what kind of data is going to be stored in the database, which makes mongo quite flexible.

3.4. Week 20 – Aggregation 1

This week's lectures focus was on aggregation. MongoDB provides three ways of aggregation: Single purpose aggregation, aggregation pipeline and Map-reduce. Single purpose aggregation include count which counts the number of entries, 'group' which groups values by a given key and distinct which returns values of a key.
Aggregation pipeline allows writing a set of operations on a given collection. Operations in the pipeline are executed in the order in which they are written.

3.4.1. Lab Tasks

The task given this week was to run a set of aggregate commands and understand how they work.

1. Return States with Populations above 10 Million

```
db.zipcodes.aggregate( [
{ $group: { _id: "$state", totalPop: { $sum: "$pop" } } }, { $match: { totalPop: { $gte:
10*1000*1000 } } } ])
```

In this command the documents from the collection zipcodes are grouped by 'state' and the population field is added which gives total population of each state. The match command matches all document whose total population is greater than 10 million.

2. Return Average City Population by State

```
db.zipcodes.aggregate( [
{ $group: { _id: { state: "$state", city: "$city" }, pop: { $sum: "$pop" } } },
{ $group: { _id: "$_id.state", avgCityPop: { $avg: "$pop" } } } ])
```

In this query, in the first group stage the documents are grouped by state and city and the population is added together. In the second group stage the documents are grouped by state and the average population is each city is calculated.

3.4.2. Weekly Reflection

The concept of Single pipeline aggregation is easy to understand since they are quite similar to the commands in SQL. The basic aggregation pipeline commands provided in the lecture and practical documents are understood, but the author has not fully understood the use of all the commands available for aggregation pipeline stages.

3.5. Week 21 – Aggregation 2

On week 21 the focus was on better understanding aggregation and accumulators were introduced as well. Accumulators are operators available in the '$group' stage of aggregation. Accumulators include operators such as '$add', '$multiply', '$divide', etc. These operators are available throughout the group stage and can be accessed using the '$group's operators ID. To understand aggregation better, some exercises were provided which are indicated in the section below.

3.5.1. Lab Tasks

1) Exercises from MongoDB-Aggregation Guide (Zipcode Exercises)

Figure 20 - Zipcode Exercise query

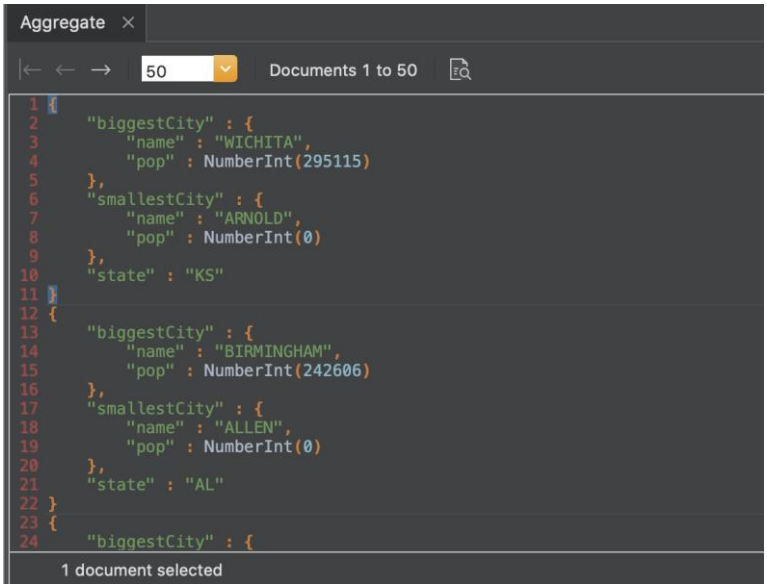


Figure 21 - Zipcode exercise output

3.5.2. Weekly Reflection

The recap of aggregation and the exercises helped the author understand the topic better. The author feels confident to write different kinds of aggregation queries.

3.6. Week 22 – Aggregation 3

The lecture this week covered topics such as set operators, string operators and Map Reduce. Map Reduce is a command for aggregation, where the map function performs operations on the input data and returns a key-value pair. This key-value pair then acts as an input for the reduce function which combines values for keys that have multiple values.

3.6.1. Lab Task

The lab task given this week was to create a collection called ‘orders’ and use map-reduce on it.

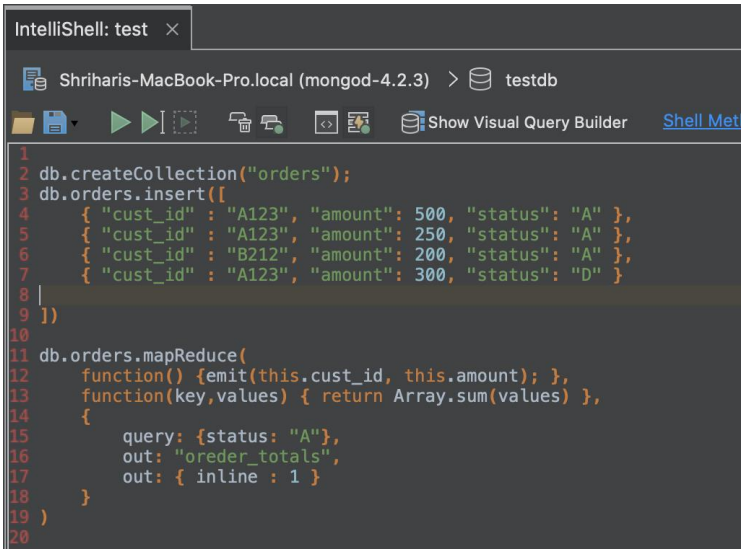


Figure 22 - Map Reduce

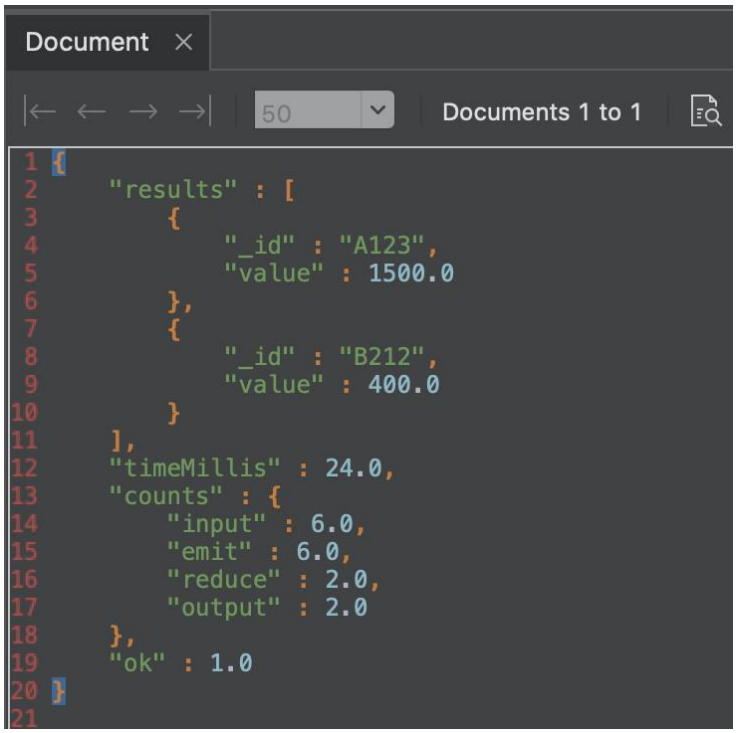


Figure 23 - Map Reduce Output

3.6.2. Weekly Reflection

This week the concept of set operators, string operators and map-reduce was understood.

4. MongoDB Database Task

A dataset containing the information of games played in the English Premier League (2018-2019) has been provided. The given task is to import the dataset, create a database and query the database using MongoDB. This section contains a description of the solution for the given task. All MongoDB tasks are carried out using Studio3T. (*Note: This section illustrates code and output using images, the code can be found in the appendix section.)*

4.1. Import data set and create database.

To import the dataset all fields and values must be compatible with MongoDB. In the given dataset the fields "BbMx>2.5", "BbAv>2.5", "BbMx<25" and "BbAv<2.5" are not valid fields. These fields are replaced to "BbMx2,5,gt", "BbAv2,5,gt", "BbMx2,5,lt" and "BbAv2,5,lt" respectively, using find and replace option available in a text editor. This makes all the fields and values compatible with MongoDB.

To import the data Studio3T's import feature is used. The dataset is imported into a database called 'assignment' and into a collection called 'games'. An illustration of the import screen is shown below:

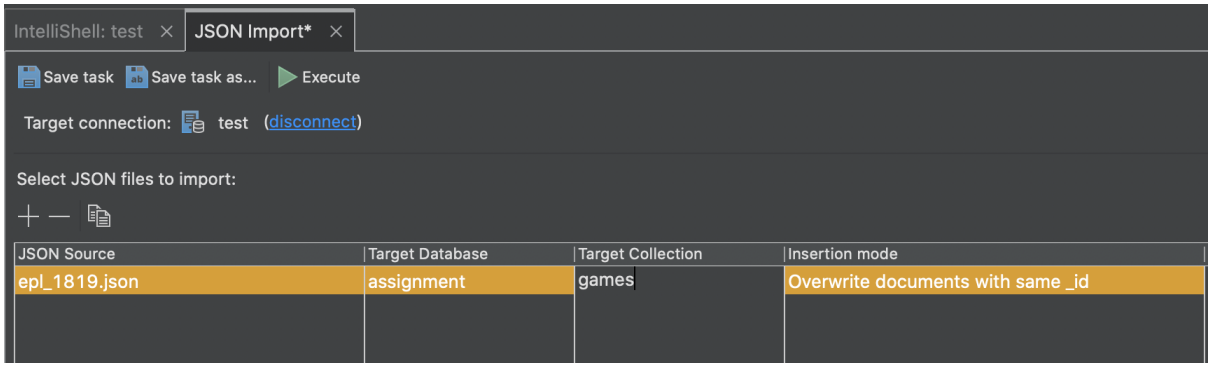


Figure 24 - Import Dataset

Once the 'execute' button is clicked the data is successfully imported. In each document, there are fields containing the betting odds for each match. These fields are not required since the given task does not contain any queries related to the betting odds. Therefore, these fields are removed using the command given below:

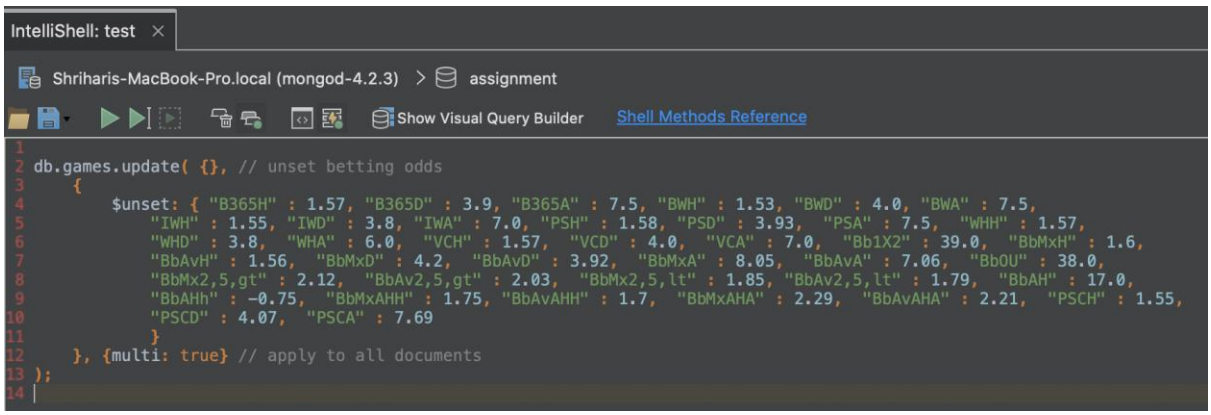


Figure 25 - Remove Betting odds

Once this query is executed, the database is ready for querying. The following image shows the output of the above command.

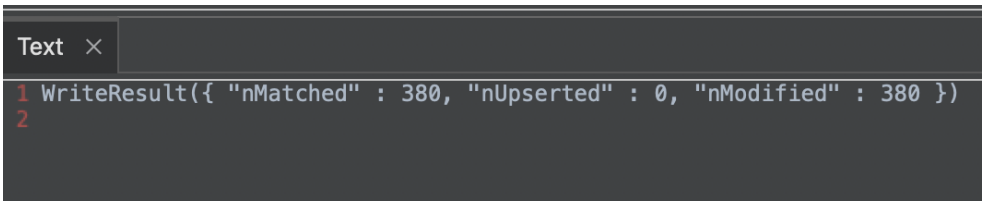


Figure 26 - Remove betting odds output

4.2. Query to display all EPL teams involved

The query to show all EPL teams can be done using the 'distinct' command. The distinct command fetches the distinct values of the specified field, this means that it will show all values once irrespective of the number of times it occurs in the collection. The following two images show the query and output.

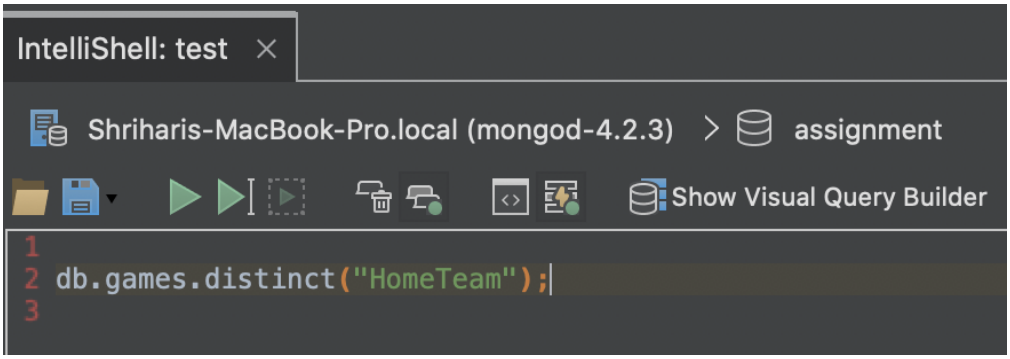


Figure 27 - Display all teams

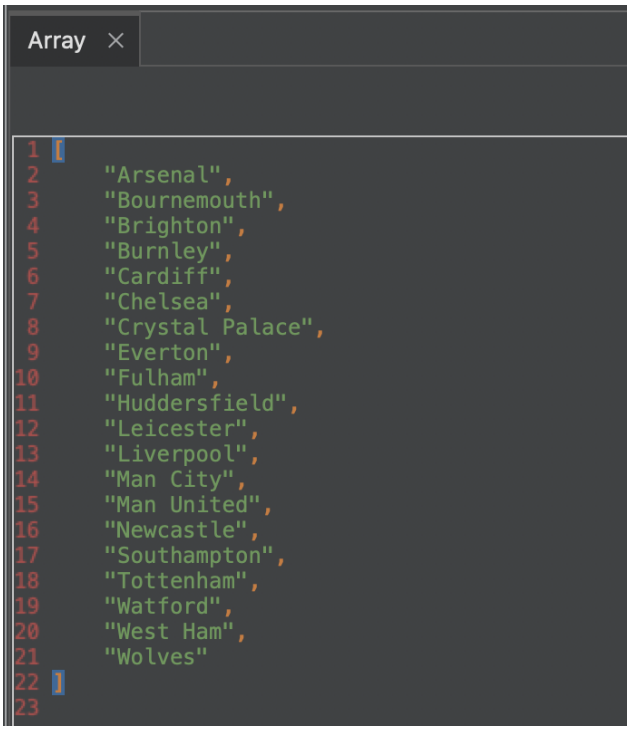


Figure 28 - Display teams - output

4.3. Query to display the number of matches played on a Monday

This query can be performed using aggregate, project, match and group. The query and output are shown below.

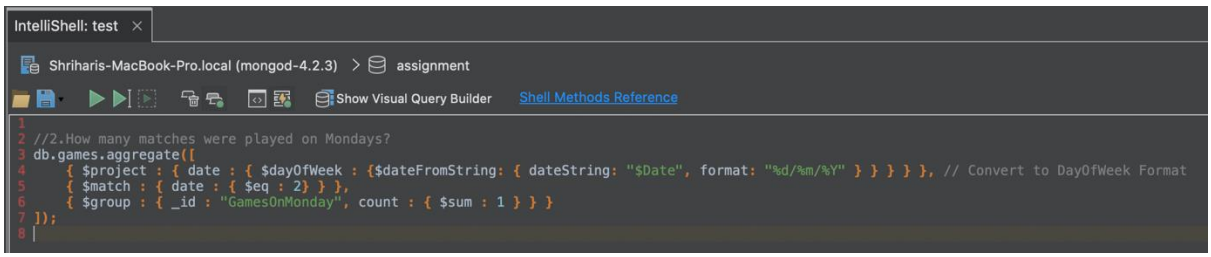


Figure 29 - Matches played on Monday

Figure 30 - Matches played on Monday Output

The MongoDB aggregate command allows a set of operations to be performed on a given collection. Each operation is executed step by step in the order they are written inside the aggerate command. '$project' command allows new fields to be calculated and passed on to the next operation in the pipeline. In the above command, '$project' is used to convert the "Date" field in the document to day of the week format. In the given dataset the "Date" field is stored as a string which is converted to date format using '$dateFromString'. This is then converted to a day of the week using '$dayOfWeek' operator. This operator returns 1 if the date is on a Sunday, 2 if on a Monday and so on. This value is then passed on to the match stage.

During the match stage all documents whose 'date' field matches the value '2' (2 represents Monday in day of the week format) are kept and the rest are discarded.

In the group stage '$sum' is used to add the number of remaining documents. This gives us the sum of documents which store information of games played on a Monday, which is the result for the number of games played on a Monday.

4.4. Query to display the total number of goals scored and conceded by Liverpool

The output for this query can be achieved by using aggregate, facet and project. The following screenshots indicate the query and the output.

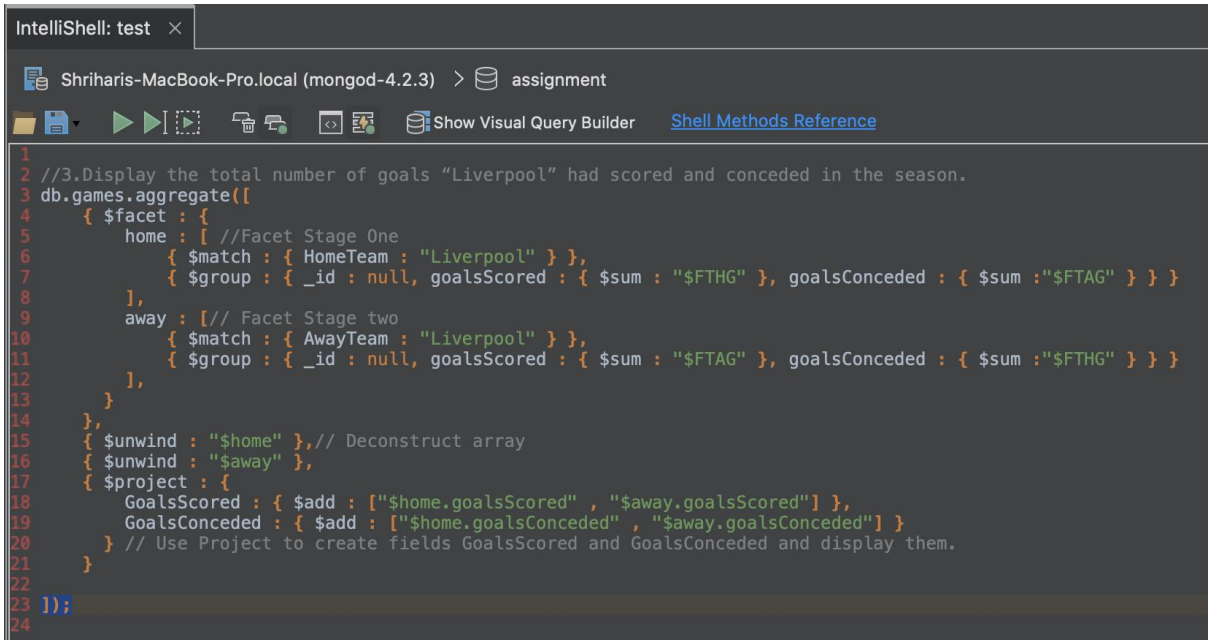


Figure 31 -Goals scored and Conceded by Liverpool

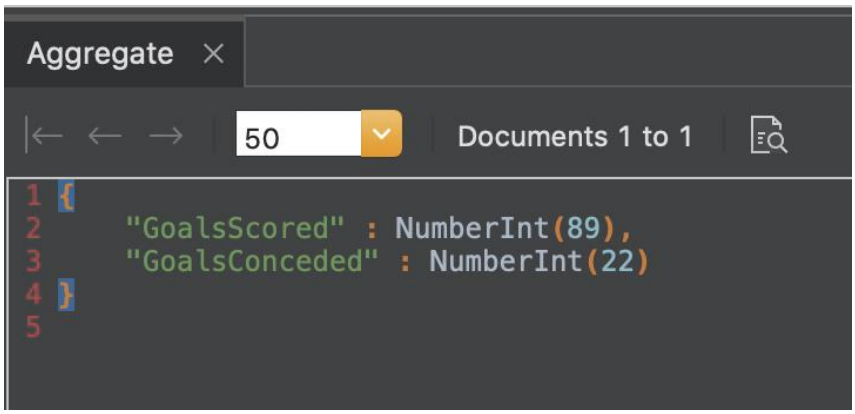


Figure 32 -Goals scored and Conceded by Liverpool, output

The aggregate command allows to group operations to perform on the input documents. '$facet' can be thought of as a set of aggregate commands where the output for each aggerate command is stored in an array. For this query, '$facet' has two sub-pipelines, 'home' and 'away'. In the 'home' stage, documents in which Liverpool is the home team is matched and the fields "FTHG" and "FTAG" are added as 'goalsScored' and 'goalsConceded' respectively. The same procedure is followed for the 'away' pipeline, but here the documents matched are the ones where Liverpool is the away team.

At this stage, 'home' is an array which contains the Goals scored and conceded by Liverpool as the home team. Similarly, 'away' contains the Goals scored and conceded by Liverpool as the away team. Since both these variables are arrays, the '$unwind' command is used to deconstruct the array into individual elements.

In the project stage adding the 'goalsScored' from 'home' and 'away' gives us the total number of goals scored by Liverpool. Similarly, the goals conceded is calculated.

When this query is executed, we get the output for Goals Scored and conceded by Liverpool in the season.

4.5. Query to display who refereed the most matches

The result for this query is achieved using an aggregate command. The query and output are illustrated below.

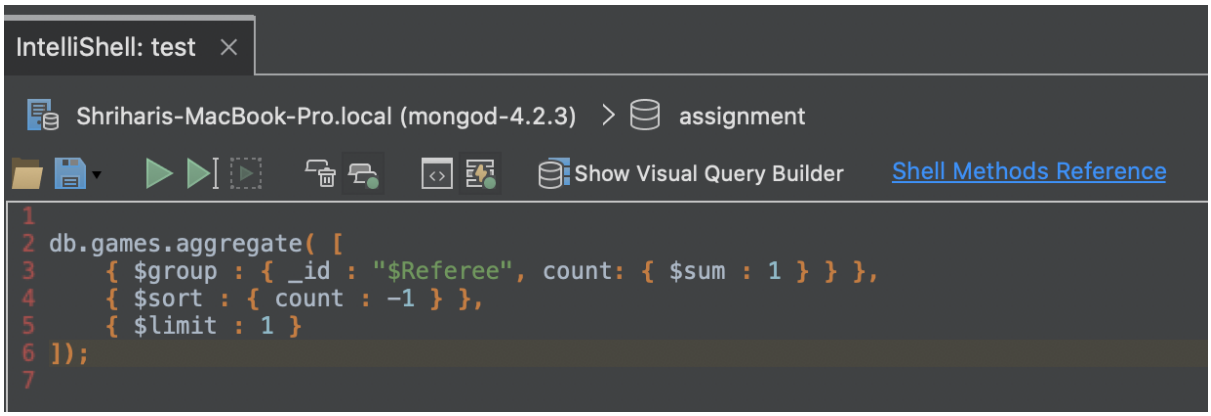


Figure 33 - Query to show Refereed most matches

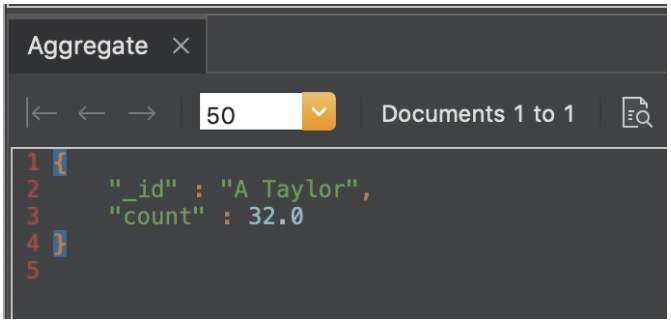


Figure 34 - - Query to show Refereed most matches output

Here the aggregate command is used to create a pipeline. The first stage of the pipeline is '$group', which groups by "$Referee", which is the field for referee name, and returns the number of times a referee appears in the collection 'games'. This output is then sorted in descending order using '$sort'. Since we only need the name of the referee who refereed most matches, the limit of the output is set to 1. This gives us the name of the referee who has refereed the most matches, and the number of matches he has refereed.

4.6. Query to Display all the matches where Man United Lost

All matches lost by Man United are displayed using a query that uses the 'find' command and some logical operators. The following images indicate the query and output.

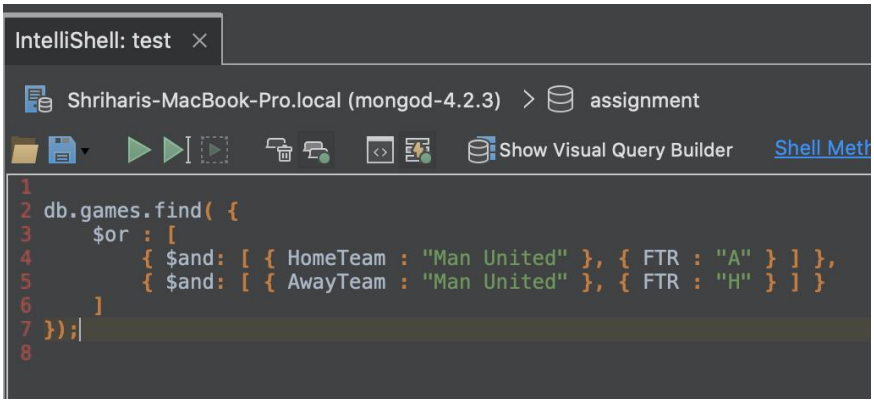
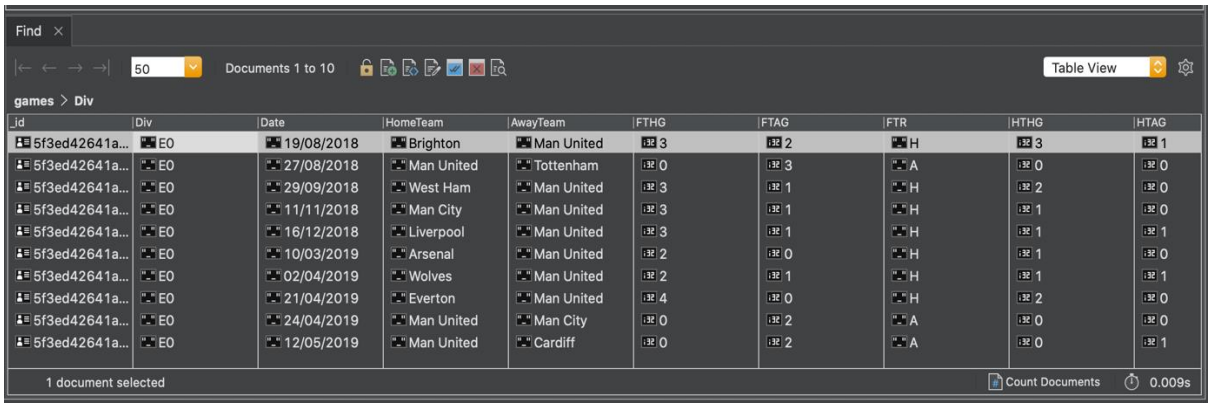


Figure 35 - Matches Man United Lost



_id	Div	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	HTHG	HTAG
5f3ed42641a...	E0	19/08/2018	Brighton	Man United	3	2	H	3	1
5f3ed42641a...	E0	27/08/2018	Man United	Tottenham	0	3	A	0	0
5f3ed42641a...	E0	29/09/2018	West Ham	Man United	3	1	H	2	0
5f3ed42641a...	E0	11/11/2018	Man City	Man United	3	1	H	1	0
5f3ed42641a...	E0	16/12/2018	Liverpool	Man United	3	1	H	1	1
5f3ed42641a...	E0	10/03/2019	Arsenal	Man United	2	0	H	1	0
5f3ed42641a...	E0	02/04/2019	Wolves	Man United	2	1	H	1	1
5f3ed42641a...	E0	21/04/2019	Everton	Man United	4	0	H	2	0
5f3ed42641a...	E0	24/04/2019	Man United	Man City	0	2	A	0	0
5f3ed42641a...	E0	12/05/2019	Man United	Cardiff	0	2	A	0	1

Figure 36 - Matches Man United Lost output

The find query displays all documents that match the given criteria. In this query, logical operators '$or' and ''$and' are used to find matches where Man United was the home team and the other team won or vice-versa. This gives us a list of matches which Man United lost.

4.7. Query to display final ranking of the teams based on their total points

This query was solved two different ways, both the methods are mentioned below.

4.7.1. Method – 1: Map Reduce

In this method, MongoDB's 'mapReduce' command is used to query the data. In map-reduce, the map function (the initial function in the pipeline) is applied to each input document. The output of a map function is a key-value pair. If multiple values are outputted for a single key, then the reduce function is applied (the second function of the pipeline). In the reduce function, multiple values associated with a single key are reduced to a single value. The images below show the query and output using Map-reduce.

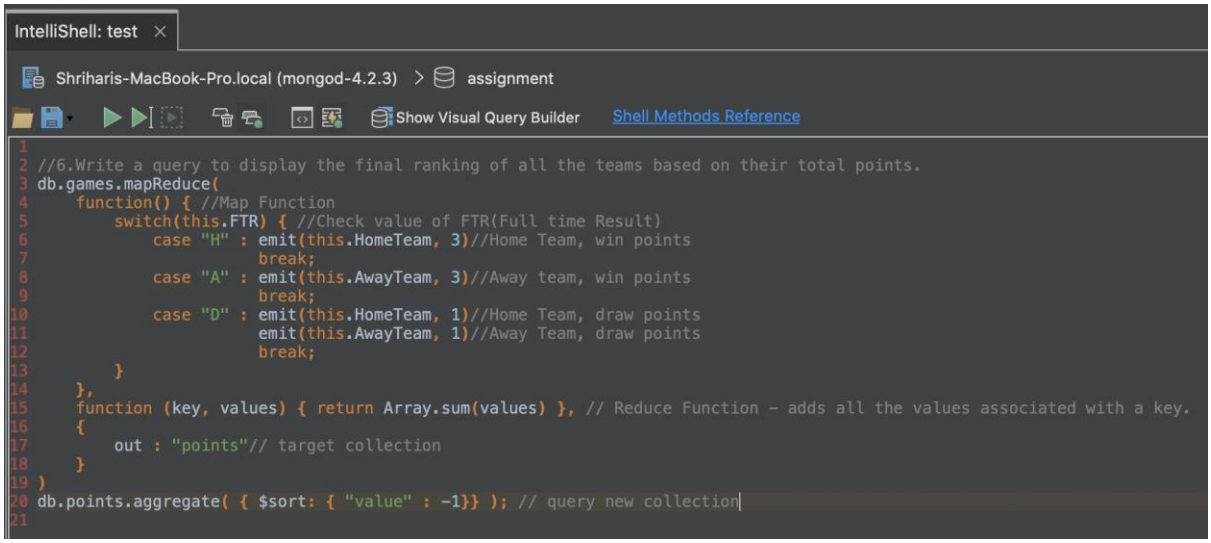


Figure 37 - Total Points - Map Reduce

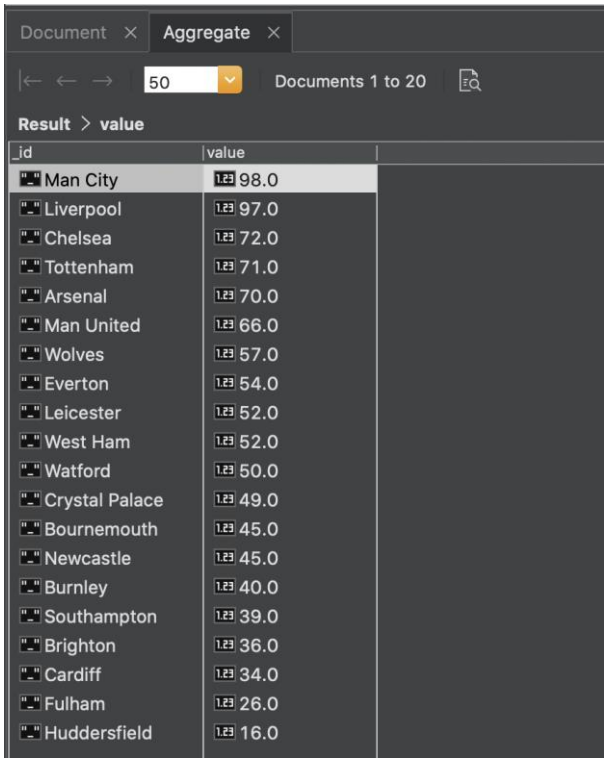


Figure 38 - Total Points - Map Reduce output

In the above query, the map function performs a switch case for the field 'FTR' which is full time result of a game. If the 'FTR' value is 'H' which means the home team won, then the emit function creates a key-value pair with key as the home team name and value as 3, which is the number of points awarded for a win. The same procedure is followed for away team. When the game is a draw, both teams are awarded a point, therefore there are two emit functions which create key-value pairs for home and away team. An example output of the map function is: *Arsenal: [3, 1, 3, 3, 3].*
Next is the reduce stage. The reduce function takes the output from the map function and adds all values for each key. Example output for this stage would be: *Arsenal: 13.*
Finally, the output from the reduce function is stored in a collection called 'points'.

Querying the 'points' collection with aggerate and sort gives us the list of teams and their total points.

4.7.2. Method – 2

This method uses some JavaScript along with MongoDB. The mongo commands used are aggregate, facet, unwind, set and project. The query and the output are given below.

IntelliShell: test

Shriharis-MacBook-Pro.local (mongod-4.2.3) > assignment

Show Visual Query Builder Shell Methods Reference

```
//6.Write a query to display the final ranking of all the teams based on their total points.
var x = String(db.games.distinct("HomeTeam"));// Get teams names as a single string
var teams = x.split(","); // Make and array of team names
var points = {}; //empty dictionary to store the points for each team

function getPoints(team){ // function  to calculate the points for one team
     var points = db.games.aggregate([
        { $facet : {
            wins : [ // sub-pipeline 1 - calculate win points
                { $match : { $or : [ {$and : [{HomeTeam : team}, {FTR : "H"}]}, {$and : [{AwayTeam : team}, {FTR : "A"}]} ]} },
                { $group : { _id : null , count: { $sum: 3 } } },
            ],
            homeDraw : [ // sub-pipeline 2 - calculate draw points as home team
                { $match : { $and : [ { HomeTeam : team }, { FTR : "D" } ] } },
                { $group : { _id : null , count: { $sum: 1 } } },
            ],
            awayDraw : [ // sub-pipeline 3 - calculate draw points as away team
                { $match : { $and : [ { AwayTeam : team }, { FTR : "D" } ] } },
                { $group : { _id : null , count: { $sum: 1 } } },
            ]

          }
        },
        { $unwind : "$wins" }, // deconstruct array
        { $unwind : { path: "$homeDraw", preserveNullAndEmptyArrays: true } }, // deconstruct array and do not discard null values
        { $unwind : { path: "$awayDraw", preserveNullAndEmptyArrays: true } }, // deconstruct array and do not discard null values
        { $set :
              {
                 "home" : { $ifNull : ["$homeDraw.count" , 0] }, //convert null values to 0
                 "away" : { $ifNull : ["$awayDraw.count" , 0] } //convert null values to 0
              }

         },
         { $project : {
                points : { $add : [ "$wins.count", "$home" , "$away" ] } // add all the points to get the total points
           }
         }

    ]).toArray(); // convert the output to Array
    return points[0]["points"]; // return the output
}

for (var team in teams){
    points[teams[team]] = getPoints(teams[team]); // calculate total points for each team
}

printjson(points); // output
```

Figure 39 - Total Points

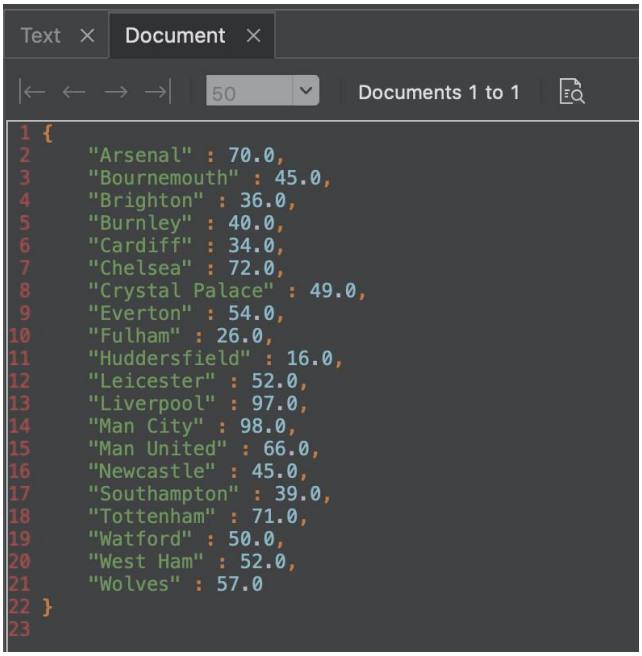


Figure 40 -Total Points Output

Query Description:

1. The variable 'x' stores the names of the teams as a single string, and the variable team stores the names of teams as an array.
2. The function 'getPoints' takes the team name as an input, calculates the points for that team and then returns the points.
3. The aggregate command is used to create a pipeline to calculate the points for a given team.
4. The '$facet' stage has three sub-pipeline, one to calculate the win points of the team, one to calculate the draw points of the team when they are the home team, and one to calculate the draw points of the team when they are the away team.
5. In the 'wins' sub-pipeline of '$facet' the documents matched are either the ones where the field 'HomeTeam' matches the given team name and they have won the match, or the ones where the field 'AwayTeam' matches the given team name and they have won the match. Once the documents are matched, the number of documents is counted and multiplied by 3, which is the points awarded for a win.
6. In the 'homeDraw' sub-pipeline, the documents matched are the ones where 'HomeTeam' is equal to the given team name and when the result is a draw. The matched documents are counted in the group stage.
7. In the 'awayDraw' sub-pipeline, the process is the same as 'homeDraw', but the documents matched are the one where 'AwayTeam' is equal to the given team.
8. '$unwind' then deconstructs all the arrays to single values and keeps values that are null where required.
9. In the next step, '$set' is used to change null values to zero.
10. Finally, in the project stage, the total points for the team is calculated by adding all the variables.
11. The 'for' loop iterates over the 'teams' variable and calls the 'getPoints' function for each team. The output of the function is stored in the 'points' variable.
12. The points variable is converted to JSON and printed to get the desired output.

5. Essay

This section contains an essay written on ‘Big Data Analytics and the role of NoSQL’. The essay is available from the next page.

Big Data Analytics and the role of NoSQL

Shrihari Muralitharan, *Student, University of Northampton*

***Abstract*— Big Data and NoSQL are common buzzwords used when talking about data. Big data is data that is large in volume, increasing rapidly and has many varieties. The word was coined when technological advancements generated a large amount of data which lacked structure. NoSQL is a group of databases which do not store data in the traditional relational format. This kind of database was made to handle big data. This paper discusses Big Data and NoSQL data stores in detail. It also covers a few background topics required to understand NoSQL and Big Data better.**

I. INTRODUCTION

Big data refers to a volume of data that is too huge to be stored and processed in a short time by traditional data management tools. Datastore exceeding a measurement of gigabyte, terabyte and petabyte is generally considered Big Data. Data in smaller measurements can also be considered Big Data in some cases. For example, 10 gigabytes of photos which need processing in some way, can be considered as Big Data for a personal computer with low processing power [1].

Big Data is generated from a variety of sources, some of which include social media, data generated from sensors, machine log data and public web data. The data generated from these sources are categorized into four main categories which are Volume, Variety, Velocity and Veracity, veracity being the more recent addition.

- Volume refers to the amount of data generated.
- Variety refers to the different kinds of data generated, some of which include media (photos, videos, GIFs), email.
- Velocity is the speed at which the data is generated or processed.
- Veracity refers to the noise in data, such as inconsistencies or missing data.

Data generated can also be classified into the following types:

- Structured Data: Data that has a well-defined structure and can be easily stored and accessed from a database fall under this category. An example of this kind of data is a table that contains information about students studying a course.
- Unstructured Data: Data that does not follow a consistent format and is hard to access or process, is classified as Unstructured Data. Some examples include image files, audio files and email.
- Semi-Structured Data: Data that includes both Structured and Unstructured data is known as Semi-Structured Data. Example of such data includes XML and JSON files.

NoSQL, the abbreviation of "Not Only SQL", is an alternative to the conventional Relational Database which stores data in tables. NoSQL is a term that refers to several methods used for storing data. These methods support horizontal scalability, flexible schema, parallel processing, and storage of complex data types, all of which are some of the drawbacks of Relational Databases. NoSQL and its features allow storing, processing and analysing Big Data simpler.

The following sections discuss Big Data and NoSQL in detail and explain the role of NoSQL in Big Data Analytics. Some topics also include the benefits of using NoSQL over Relational Databases.

II. BACKGROUND CONCEPTS

The following section covers a few concepts, in brief, that is required to understand the rest of this project.

A. Relational Database, Schema and SQL

Relational Databases is known as the traditional method of storing data.
In this approach, data is stored in rows and columns which form tables, a collection of tables which have a relationship with each other form a database. Each table has a primary key and may have a foreign key which is used to interact with other tables.
A schema acts as a blueprint for a database. It describes the construction of a database and the associated objects. A schema also describes the kind of relationship the objects would maintain with each other.
Structured Query Language or SQL is the language used to access and modify the contents of a relational database.

Examples of Relational Database Management Systems (RDBMS): Oracle Database, MySQL, Microsoft SQL Server and PostgreSQL.

B. Scalability, Distributed Systems, Parallel Processing and Data Replication

Scalability refers to the ability of a system to meet changing demands by adding or removing resources. The following are the two types of Scalability:

1. Vertical Scalability or Scale-up: This involves adding computing power and resources to a single server. This is quite expensive and is impractical after a point.
2. Horizontal Scalability or Scale-out: This involves the use of distributed systems which is an approach where data is stored across multiple servers. Each server is known as a node and a cluster of nodes form a Distributed System. It is cheaper to add a new server than to add resources to an existing server. Thus, it is more practical to store data that is increasing at an exponential rate in a Horizontal Scalable system. Horizontal scaling is also referred to as sharding or partitioning

Parallel Processing is when a process is split into multiple parts which are processed simultaneously on different processors or servers.

Data Replication is essentially creating copies of data in a different node/system. Replication improves reliability and makes data available at all times.

C. CAP Theorem

CAP theorem or Brewer's theorem states that for a distributed system, only two out of the three, Consistency, Availability or Partition Tolerance, can be maintained. The following is a brief description of these properties:

- Consistency: A distributed system is said to have consistency if all nodes that access the data can access the same data. This means that a transaction (a set of commands) is processed only if all the commands in the transaction are executed without any error. When a system is consistent, any changes (such as the addition of data) must be updated on all nodes before the nodes are available. For example, when two people are working on some data from the same database at the same time, if one person adds or modifies the data then the other person will not be able to make any changes to the data until the first persons' operation is complete. And if the first person issues a set of commands, the operation will fail even if one command fails.
- Availability: Availability means that a request for data always gets a response regardless of the state of an individual node. This means that data must be accessible even if a node/server is down.
- Partition Tolerance: A partition is a communication delay or breakdown between two nodes. A partition tolerant system continues to work regardless of the delay or break between the nodes. To achieve this, data from the nodes is replicated onto other nodes or networks. All the nodes are synched later to achieve eventual consistency.

Most NoSQL data stores focus on Availability and Partition at the cost of consistency. NoSQL data sores follow the BASE principle, which is an acronym for Basically Available, Soft State, Eventually Consistent [3].

SQL follows the principle of ACID, which is covered in detail in the next section.

D. Transactions and ACID

A transaction is a logical, atomic unit of work that contains one or more SQL statements [2]. A transaction groups SQL statements so that they are either all committed, which means they are applied to the database, or all rolled back, which means they are undone from the database [2]. All database transactions have the following properties:

- Atomicity: Either all or none of the tasks of a transaction are performed. There are no partial transactions. For example, if a transaction starts updating 100 rows, but the system fails after 20 updates, then the database rolls back the changes to these 20 rows [2].
- Consistency: The transaction takes the database from one consistent state to another. For example, in a banking transaction that debits a savings account and credits a checking account, a failure must not cause the database to credit only one account, which would lead to inconsistent data [2].
- Isolation: The effect of a transaction is not visible to other transactions until the transaction is committed. For example, one user updating the hr.employees table does not see the uncommitted changes to employees made concurrently by another user. Thus, it appears to users as if transactions are executing serially [2].
- Durability: Changes made by committed transactions are permanent. After a transaction completes, the database ensures through its recovery mechanism that changes from the transaction are not lost [2].

The above four properties are referred to as ACID.

III. Types of NoSQL Databases

NoSQL refers to a variety of models which use methods other than the traditional relational model to store data. NoSQL models were put together to address the

drawbacks present in the Relational Model. The following are the four major types of NoSQL Databases:

A. *Key-Value Stores*

This is a simple kind of data store which is mainly based on Amazon's Dynamo. In this kind of data store, hash tables/dictionaries are used as the main data model. A hash table has two elements, a key and a value. Each key is unique, and it stores the address of a specific value (essentially a pointer). This kind of datastore does not have a specific query language, it uses methods such as get, put, delete and execute to add and modify data.
Example of Key-Value Databases: Amazon Dynamo, Redis, Memcached and Voldermort.

B. *Wide-Column Stores*

In this approach, data is stored in columns instead of rows. These tables are similar to relational tables but can store semi-structured data. Scalability is achieved in this model by splitting both rows and columns over multiple nodes.
Examples of Wide-Column Stores: Cassandra, BigTable and HBase.

C. *Document Stores*

Data is stored in JSON, XML or BSON like file formats. Each document contains pairs of keys and values. The values can store several types of data including complex data structures. Documents in this database do not follow a schema and do not support Joins.
Example of Document Store: MongoDB and CouchDB

D. *Graph Stores*

In this approach, data is stored in nodes and edges. Nodes contain the data and edges contain the relationship between nodes. This approach is inspired by Mathematical Graph Theory and is most useful when there is a need to traverse relationships to identify patterns.
Examples of Graph Stores: Neo4j.

IV. Big Data and NoSQL

Big Data, as discussed earlier, is huge volumes of data that is difficult to store, process and analyse. Research indicates that the volume of data doubles every two years, and this data comes in different and new varieties. Big Data Analytics is the process used to extract meaningful information, find patterns and visualize Big Data. Big data analytics helps to understand and get insights from data which can be used to build a better business model, tech companies such as Google, Amazon and Facebook are known to take advantage of data analytics.
For example, Amazon, one of the biggest e-commerce websites, collects data such as customer purchases, interests, etc. and analyses this data to provide product suggestions. The data collected by amazon is considered Big Data since it has an enormous amount of customer.

A. *Streaming Data*

Data that is generated continuously from various sources is known as Streaming Data. Examples of such data include log files, data generated from sensors and IoT (Internet of Things) devices. Streaming data is generated at a constant rate and does not have a limit. The generated data is constantly analysed, and insights are gained over time. Streaming data is also known as streaming analytics, real-time analytics, real-time streaming analysis and event processing. Streaming data is considered as big data since the data generated is large in volume. Frameworks like Apache Kafka, Apache Flink (discussed later) are used for processing Streaming Data and NoSQL acts as a great database to store Streaming data. NoSQL data stores are a great option since the Streaming data is unstructured and is hard to store in Relational Databases.

B. *Map Reduce*

Map reduce is a programming model which is used to process data in the shard/node where the data is stored. This makes the processing of large volumes of data very fast. Some NoSQL databases like MongoDB support Map Reduce.
Map Reduce consists of two separate functions 'Map' and 'Reduce'. 'Map' is responsible for breaking down data into tuples (key, value pairs). 'Reduce' takes the input from the 'Map' function, processes the data and produces the output.

C. *NoSQL vs Relational Databases for Big Data*

1. Horizontally Scalable: NoSQL data stores are made for horizontal scalability. Horizontal scalability is essential for Big Data, which makes NoSQL a better option than Relational Databases which are known to work best with vertical scalability. Some NoSQL databases like MongoDB provide built-in auto-sharding, making it very simple to use.
2. Lack of Schema: NoSQL databases do not require a schema or use a very lenient schema. This means that the data models used are flexible and can be changed when required. This kind of flexibility is not possible in Relational databases since the schema needs to be designed in the early stages.
3. Data structures: NoSQL provides support for complex data structures, making it easier to store both structured and unstructured data. Relational databases either lack or provide minimal support for complex data structures.
4. Database Property: Most NoSQL databases follow the BASE principle, which means that the focus is on availability and partitioning and not on consistency. NoSQL does not

compromise consistency, but they provide it eventually. Relational databases follow ACID which mainly focuses on consistency, making a very rigid data model.

5. Replication: Most NoSQL databases have built-in support for replication, MongoDB and Cassandra are few examples of databases which make replication quite simple.

D. *Disadvantages of NoSQL*

NoSQL data stores also have a few disadvantages, which are as follows:

1. Lacks data consistency or provides eventual consistency, which can be a big issue in some cases.
2. Does not support or offers minimal SQL support, which is an issue if an existing SQL database needs to be converted to NoSQL.
3. NoSQL is relatively new, therefore there is less support and stability in some cases.
4. NoSQL is not a single type of database, some of which are discussed above sections; hence it lacks a unified query language.

E. *Use Cases of NoSQL in Industry*

- Netflix - Netflix, one of the biggest video streaming platforms, originally launched its service in 2007 with an Oracle Database. Once Netflix started expanding to different countries the demand for the platform grew vastly. The demand made the company switch from Oracle Database to Cassandra (a NoSQL database).
- Facebook - In 2010 Facebook considered both SQL and NoSQL databases for its new messaging system. Facebook chose a NoSQL database since it expected high traffic.
- Nokia - Nokia a mobile phone company switched from a Relational database to a NoSQL database to avoid significant investment in upgrading the Relational Database.

V. Big Data Frameworks

The following are some of the tools, excluding NoSQL databases, used to handle big data:

A. *Hadoop*

The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models [4]. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage [4]. Hadoop consists of three main components:

1. HDFS - Hadoop Distributed Filesystem which is responsible for storing data in Hadoop clusters.
2. MapReduce - allows parallel processing of large volumes of data.
3. YARN - Allows batch, stem, interactive and graph processing of data. The main task of YARN is to split the functionality of resource management and job scheduling/monitoring.

B. *Spark*

Apache Spark is a general-purpose distributed data processing engine. The difference between spark and Hadoop is that spark uses in-memory processing which makes it faster. Spark lacks a file system and must use other storage systems like HDFS. Spark supports multiple programming languages like Python, R, Scala and Java.

C. *Hive*

Apache Hive is a data warehouse system built on top of Hadoop. Hive allows developers to query data in a standard that is similar to SQL. Hive can convert queries written in Hive Query Language (HQL) to MapReduce.

D. *Storm*

Apache Storm is a distributed real-time computing system. Storm is great for handling continuous data that has an unbound limit. Apache Storm uses storm clusters which use Topologies instead of MapReduce. MapReduce eventually ends, whereas topology processes go on until the process is killed.

E. *Kafka*

Apache Kafka is a stream-processing platform which provides high throughput, low-latency for handling streaming data. Streaming data, as discussed earlier, is data that is generated continuously. Kafka combines messaging, storage and stream processing real-time data.

VI. Conclusion

Data is growing at an exponential rate with the increase in the speed of technological advancements and human dependency on technology. Traditional methods like Relational Databases are outdated and have many drawbacks, thus they are not well suited to store Big Data.

NoSQL is a group of new and upcoming databases that use different methods to store data. NoSQL data stores have the following characteristics:

1. Can store structured, semi-structured and unstructured data.
2. Support horizontal partitioning, which makes it cheaper and easier to store large amounts of data.
3. Have a flexible schema which allows changing of data model when required.
4. Allow storage of complex data structures.
5. Use BASE properties which are more lenient than ACID.

These characteristics that NoSQL databases are some of the requirements for Big Data, which makes NoSQL a great candidate for Big Data.
NoSQL works well in most cases, but in some cases, Relational databases might be better to use. For example, any data that is small in volume, doesn't increase fast and can easily be stored in rows and columns or excel like spreadsheets might work better if stored in Relational Databases.

II. ACKNOWLEDGEMENT

I would like to thank my lecturer Dr. James Xue for providing me with an opportunity to write this paper. I would also like to thank him for the support he has provided for writing this paper.

VII. REFERENCES

[1] Shahzan, 2019. *Big Data Explained in Plain Simple English.* [Online]
Available at: https://medium.com/swlh/big-data-explained-38656c70d15d
[Accessed August 2020].

[2] Oracle, n.d. *Transactions.* [Online]
Available at: https://docs.oracle.com/database/121/CNCPT/transact.htm#CNCPT016
[Accessed 2020].

[3] Chandra, D. G., 2015. BASE analysis of NoSQL database. *Future Generation Computer Systems,* Volume 52, pp. 13-21.

[4] Apache Hadoop, n.d. *Apache Hadoop.* [Online]
Available at: https://hadoop.apache.org
[Accessed 2020].

[5] Jose, B. & Abraham, S., 2020. Performance analysis of NoSQL and relational databases with MongoDB and MySQL. *Materials Today: Proceedings,* Volume 24, pp. 2036-2043.

[6] Moniruzzaman, A. B. M. & Hossain, S. A., 2013. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. *International Journal of Database Theory and Application,* 6(4).

[7] Kambatla, K., Kollias, G., Kumar, V. & Grama, A., 2014. Trends in big data analytics. *Journal of Parallel and Distributed Computing,* 74(7), pp. 2561-2573.

[8] Corbellini, A. et al., 2017. Persisting big-data: The NoSQL landscape. *Information Systems,* Volume 63, pp. 1-23.

[9] Kanwar, R., Trivedi, P. & Singh, K., 2013. NoSQL, a Solution for Distributed Database Management System. *International Journal of Computer Applications,* 67(2).

[10] Kunda, D. & Phiri, H., 2017. A Comparative Study of NoSQL and Relational Database. *ZAMBIA INFORMATION COMMUNICATION TECHNOLOGY (ICT) JOURNAL,* 1(1), pp. 1-4.

[11] Venkatraman, S., Kiran Fahd, S. K. & Venkatraman, R., 2016. SQL Versus NoSQL Movement with Big Data Analytics. *I.J. Information Technology and Computer Science,* Volume 12, pp. 59-66.

[12] Amazon AWS, n.d. *What is Streaming Data.* [Online]
Available at: https://aws.amazon.com/streaming-data/
[Accessed 2020].

[13] Perera, S., 2018. *A Gentle Intoduction to Streaming Data.* [Online]

[14] Tan, K.-L., 2009. *Distributed Database Systems.* [Online]
Available at: https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-39940-9_701
[Accessed 2020].

[15] Nazrul, S. S., 2018. *CAP Theorem and Distributed Database Management Systems.* [Online]
Available at: https://towardsdatascience.com/cap-theorem-and-distributed-database-management-systems-5c2be977950e
[Accessed 2020].

[16] IBM Cloud Education, 2019. *What is Cap Theorem.* [Online]
Available at: https://www.ibm.com/cloud/learn/cap-theorem
[Accessed 2020].

[17] Apace Storm, n.d. *Storm Concepts.* [Online]
Available at: https://storm.apache.org/releases/current/Concepts.html
[Accessed 2020].

[18] Le, J., 2019. *An Introduction to Big Data: NoSQL.* [Online]
Available at: https://medium.com/cracking-the-data-science-interview/an-introduction-to-big-data-nosql-96b882f35e50
[Accessed 2020].

6. References

Anderson, B. & Nicholson, B., 2020. *SQL vs NoSQL Databases.* [Online]
Available at: https://www.ibm.com/cloud/blog/sql-vs-nosql
[Accessed 2020].
Gaitonde, A., 2019. *NoSQL databases — An Introduction.* [Online]
Available at: https://medium.com/analytics-vidhya/no-sql-databases-an-introduction-eb9706fbe3
[Accessed 2020].
Le, J., 2019. *An Introduction to Big Data: NoSQL.* [Online]
Available at: https://medium.com/cracking-the-data-science-interview/an-introduction-to-big-data-nosql-96b882f35e50
[Accessed 2020].
MongoDB, n.d. *MongoDB Manual.* [Online]
Available at: https://docs.mongodb.com/manual/
[Accessed 2020].
MongoDB, n.d. *NoSQL vs SQL.* [Online]
Available at: https://www.mongodb.com/nosql-explained/nosql-vs-sql
[Accessed 2020].
MongoDB, n.d. *What is NoSQL? NoSQL explained.* [Online]
Available at: https://www.mongodb.com/nosql-explained
[Accessed 2020].
Olivera, L., 2019. *Everything you need to know about NoSQL databases.* [Online]
Available at: https://dev.to/lmolivera/everything-you-need-to-know-about-nosql-databases-3o3h
[Accessed 2020].
Studio3T, n.d. *Studio3T Documentation.* [Online]
Available at: https://studio3t.com/knowledge-base/categories/documentation/
[Accessed 2020].
Xue, J., 2020. *Week 18 Restaurant Exercise.* Northampton: s.n.

7. Appendix

7.1. Week – 18 Lab Task Code

```
// 1
db.restaurants.find().pretty()
//2
db.restaurants.find({}, {"restaurant_id":1, "name":1, "borough":1, "cuisine":1}).pretty()
//3
db.restaurants.find({}, {"_id":0, "restaurant_id":1, "name":1, "borough":1,
"cuisine":1}).pretty()
//4
db.restaurants.find({}, {"_id":0, "restaurant_id":1, "name":1, "borough":1,
"zipcode":1}).pretty()
//5
db.restaurants.find({"borough":"Bronx"}).pretty()
//6
db.restaurants.find({"borough":"Bronx"}).limit(5).pretty()
//7
db.restaurants.find({"borough":"Bronx"}).skip(5).pretty()
//8
db.restaurants.find({"grades.score": {$gt:90}}).pretty()
//9
db.restaurants.find({"grades.score": {$gt:80,$lt:100}}).pretty()
//10
db.restaurants.find({"address.coord.0": {$lt:-95.754168} } ).pretty()
//25
db.restaurants.find().sort({"name":1}).pretty()
//26
db.restaurants.find().sort({"name":-1}).pretty()
//27
db.restaurants.find().sort({"cuisine":1},{"borough":-1}).pretty()
```

7.2. MongoDB Database Task Code

```
//Delete Betting Odds
db.games.update( {}, // unset betting odds
   {
      $unset: { "B365H" : 1.57, "B365D" : 3.9, "B365A" : 7.5, "BWH" : 1.53, "BWD" : 4.0,
"BWA" : 7.5,
         "IWH" : 1.55, "IWD" : 3.8, "IWA" : 7.0, "PSH" : 1.58, "PSD" : 3.93,  "PSA" : 7.5,  "WHH"
: 1.57,
         "WHD" : 3.8,  "WHA" : 6.0,  "VCH" : 1.57,  "VCD" : 4.0,  "VCA" : 7.0,  "Bb1X2" : 39.0,
"BbMxH" : 1.6,
```

```
        "BbAvH" : 1.56,  "BbMxD" : 4.2,  "BbAvD" : 3.92,  "BbMxA" : 8.05,  "BbAvA" : 7.06,
"BbOU" : 38.0,
        "BbMx2,5,gt" : 2.12,  "BbAv2,5,gt" : 2.03,  "BbMx2,5,lt" : 1.85, "BbAv2,5,lt" : 1.79,
"BbAH" : 17.0,
        "BbAHh" : -0.75,  "BbMxAHH" : 1.75, "BbAvAHH" : 1.7,  "BbMxAHA" : 2.29,
"BbAvAHA" : 2.21,  "PSCH" : 1.55,
        "PSCD" : 4.07,  "PSCA" : 7.69
        }
   }, {multi: true} // apply to all documents
);

//1.Show all the EPL teams involved in the season.
db.games.distinct("HomeTeam");

//2.How many matches were played on Mondays?
db.games.aggregate([
   { $project : { date : { $dayOfWeek : {$dateFromString: { dateString: "$Date", format:
"%d/%m/%Y" } } } }, // Convert to DayOfWeek Format
   { $match : { date : { $eq : 2} } },
   { $group : { _id : "GamesOnMonday", count : { $sum : 1 } } }
]);

//3.Display the total number of goals “Liverpool” had scored and conceded in the season.
db.games.aggregate([
   { $facet : {
      home : [ //Facet Stage One
         { $match : { HomeTeam : "Liverpool" } },
         { $group : { _id : null, goalsScored : { $sum : "$FTHG" }, goalsConceded : { $sum
:"$FTAG" } } }
      ],
      away : [// Facet Stage two
         { $match : { AwayTeam : "Liverpool" } },
         { $group : { _id : null, goalsScored : { $sum : "$FTAG" }, goalsConceded : { $sum
:"$FTHG" } } }
      ],
     }
   },
   { $unwind : "$home" },// Deconstruct array
   { $unwind : "$away" },
   { $project : {
      GoalsScored : { $add : ["$home.goalsScored" , "$away.goalsScored"] },
      GoalsConceded : { $add : ["$home.goalsConceded" , "$away.goalsConceded"] }
     } // Use Project to create fields GoalsScored and GoalsConceded and display them.
   }

]);
```

```
//4.Who refereed the most matches?
db.games.aggregate( [
   { $group : { _id : "$Referee", count: { $sum : 1 } } },
   { $sort : { count : -1 } },
   { $limit : 1 }
]);

//5.Display all the matches that “Man United” lost.
db.games.find( {
   $or : [
      { $and: [ { HomeTeam : "Man United" }, { FTR : "A" } ] },
      { $and: [ { AwayTeam : "Man United" }, { FTR : "H" } ] }
   ]
});

//6.Write a query to display the final ranking of all the teams based on their total points.
db.games.mapReduce(
   function() { //Map Function
      switch(this.FTR) { //Check value of FTR(Full time Result)
         case "H" : emit(this.HomeTeam, 3)//Home Team, win points
                  break;
         case "A" : emit(this.AwayTeam, 3)//Away team, win points
                  break;
         case "D" : emit(this.HomeTeam, 1)//Home Team, draw points
                  emit(this.AwayTeam, 1)//Away Team, draw points
                  break;
      }
   },
   function (key, values) { return Array.sum(values) }, // Reduce Function - adds all the values
associated with a key.
   {
      out : "points"// target collection
   }
)
db.points.aggregate( { $sort: { "value" : -1}} ); // query new collection

// other method

var x = String(db.games.distinct("HomeTeam"));// Get teams names as a single string
var teams = x.split(","); // Make and array of team names
var points = {}; //empty dictionary to store the points for each team

function getPoints(team){ // function  to calculate the points for one team
    var points = db.games.aggregate([
      { $facet : {
         wins : [ // sub-pipeline 1 - calculate win points
```

```
            { $match : { $or : [ {$and : [{HomeTeam : team}, {FTR : "H"}]}, {$and : [{AwayTeam :
team}, {FTR : "A"}]} ]} },
            { $group : { _id : null , count: { $sum: 3 } } },
          ],
          homeDraw : [ // sub-pipeline 2 - calculate draw points as home team
            { $match : { $and : [ { HomeTeam : team }, { FTR : "D" } ] } },
            { $group : { _id : null , count: { $sum: 1 } } },
          ],
          awayDraw : [ // sub-pipeline 3 - calculate draw points as away team
            { $match : { $and : [ { AwayTeam : team }, { FTR : "D" } ] } },
            { $group : { _id : null , count: { $sum: 1 } } },
          ]

        }
      },
      { $unwind : "$wins" }, // deconstruct array
      { $unwind : { path: "$homeDraw", preserveNullAndEmptyArrays: true } }, // deconstruct
array and do not discard null values
      { $unwind : { path: "$awayDraw", preserveNullAndEmptyArrays: true } }, // deconstruct
array and do not discard null values
      { $set :
          {
            "home" : { $ifNull : ["$homeDraw.count" , 0] }, //convert null values to 0
            "away" : { $ifNull : ["$awayDraw.count" , 0] } //convert null values to 0
          }

      },
      { $project : {
            points : { $add : [ "$wins.count", "$home" , "$away" ] } // add all the points to get
the total points
          }
      }

  ]).toArray(); // convert the output to Array
  return points[0]["points"]; // return the output
}

for (var team in teams){
  points[teams[team]] = getPoints(teams[team]); // calculate total points for each team
}

printjson(points); // output
```