# Advanced Databases - MotoGP CouchDB Project

Submitted By: Darragh Elbel T00211193

Computing with Software Development

Date Submitted: dd/mm/yyyy

## Summary:

## Content:

## Introduction:

## Sections:

### Dataset

For my dataset, I chose “Moto GP World Championship(1949-2022)” available at the following link <https://www.kaggle.com/datasets/alrizacelk/moto-gp-world-championship19492022?select=riders-finishing-positions.csv> . The original data consisted of 6 separate CSV files.

The first step I took towards importing this data into CouchDB was to process the original CSV file into JSON, to do this I looped through all the files in my CSV directory and parsed them into JSON using the ‘csv-parser’ npm module.



Figure Script for parsing a directy of CSV files to JSON

After reviewing the parsed data I decided it would be best to focus on further converting one of the six CSV files into JSON, ‘riders-finishing-positions.json’. The original CSV file contained a row for each rider, a data shape that would not be optimized for a document database such as CouchDB. To remedy this I altered the data shape to have a unique object for each country code, and inserted an array of riders into this object.



Figure Updated Data Shape

To automate this alteration of data shape I wrote the following script which performs a reduce across the dataset, once complete the data is converted to JSON using JSON.stringify() and is saved to a file called ‘output.json’. It is worth noting that as this point all countries are now contained within that JSON document, and further processing is required to split these into individual documents before importing into CouchDB.



Figure Data Reshaping Script

I then split each country into its own JSON document before importing into my CouchDB database.

fs.readFile("output.json", "utf8", (err, data) => {

  const outputData = JSON.parse(data);

  outputData.forEach((countryData) => {

    const country = countryData.Country;

    const filename = `${country}.json`;

    fs.writeFile(

      filename,

      JSON.stringify(countryData, null, 2),

      "utf8",

      (err) => {

        if (err) {

          console.error(`Error writing ${filename}:`, err);

        } else {

          console.log(`Created ${filename}`);

        }

      }

    );

  });

});

After this splitting I looped through each of the 21 unique files and using a library called “node-couchdb” sent a put request for each file and added them to the database, a subset of these documents can be seen within Fauxton below.

A screenshot of a phone

Description automatically generated

CRUD Operations:

Create:

A screenshot of a computer program

Description automatically generated

Read:

A screenshot of a computer

Description automatically generated

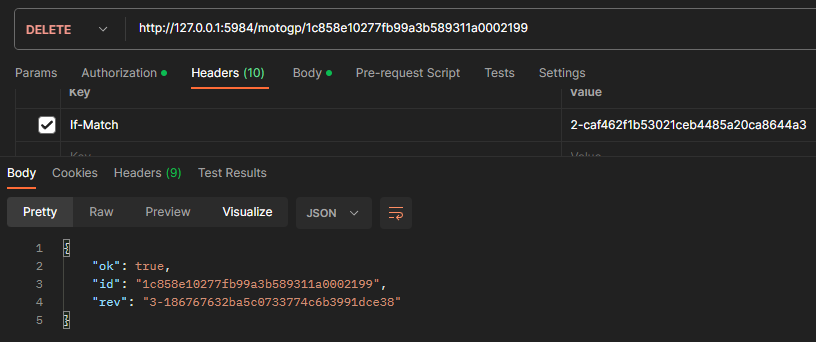
Figure Retrieving Document with GET Request

Update:

A screenshot of a computer program

Description automatically generated

Delete:



### MapReduce Functions

#### Find one particular document:

function(doc) {

  if (doc.Country === "IT") {

    emit(doc.Country, doc);

  }

}

The above function was used to find a specific document whose “Country” key had a value of “IT” for Italy. This view works by looping through each document and performing a check to see if the document’s “Country” key is equal to “IT”, if so, the view emits the identified key along with the rest of the matching document.

#### Find documents in a range:

function(doc) {

  if (doc.Country && doc.Riders && Array.isArray(doc.Riders)) {

    var totalVictories = 0;

    doc.Riders.forEach(function(rider) {

      if (rider.Victories) {

        totalVictories += parseInt(rider.Victories, 10);

      }

    });

    if (totalVictories > 100) {

    emit(null, doc);

   }

  }

}

The purpose of this function is to find return all countries whose riders have achieved a combined number of victories (1st place finishes) over 100. First, the if statement on line two verifies that the Country, Riders keys are present and that Riders is a valid array. Then a foreach loop is run to count the total number of victories, if the number is > than 100 that document is emitted. It is worth noting we are performing a parseInt on the number of victories, this is due to all numbers being stringified when I converted from CSV to JSON.

#### Use and explain the \_sum, \_count and \_stats reduce functions

##### \_sum



The map above loops through the riders array for each country, then for each rider the Country key and their number of victories is emitted. The \_sum reduce function then adds the number of victories for each like country code together.

A screenshot of a phone

Description automatically generated

Figure Output for 'riderVictoriesPerCountry' in Fauxton

##### \_count



The map above loops through the riders array for each country, then emits the country code along with the number one, the \_count function then totals up the number of riders for each country. An output of this function can be seen below.

A screenshot of a phone

Description automatically generated

Figure Output of the 'ridersPerCountry' view in Fauxton

##### \_stats



The \_stats reduce function generates statistics on the sum, count, minimum, maximum, and sum squared of each row output by the map function. An example of this output when run on my dataset can be seen below.



Figure Output of 'riderVictoriesPerCountryStats' view with \_stats reduce

##### Group and group\_level



Figure Map Function



Figure Reduce Function

### Mango Queries

#### Finding a particular document

The mango query in the following image was used to find a document by country code.

A screenshot of a computer

Description automatically generated

#### Finding documents in range

A screenshot of a computer

Description automatically generated

Mango queries allow for the use of regular expressions, in the above example all documents with a country code beginning with ‘I’ were returned.

### Java or other client for CouchDB

### Replication and mobile app using JavaScript

### DBaaS investigation

## Conclusions:

## References:

## Bibliography: