Indholdsfortegnelse

[Introduktion 2](#_Toc342652099)

[Find the ID of the film "The Rock" from 1996? (Use find()): 2](#_Toc342652100)

[Find how many 5-star ratings "The Rock" has received received? (Use find()) 2](#_Toc342652101)

[Calculate a document/cursor describing the distribution of ratings of The Rock? (That is, how many 1, 2, 3, 4, 5 star ratings?) (Use mapReduce()) 2](#_Toc342652102)

[Calculate the average rating of The Rock? (Use mapReduce(). You may calculate it on either the original ratings collection or on the output collection from the previous mapReduce.) [I get 3.723] 2](#_Toc342652103)

[Count how many movies are classifed as 'Animation' movies? 3](#_Toc342652104)

[Generate a collection "drama" containing all titles of movies classified as genre Drama. 3](#_Toc342652105)

[How many users have profession 'writer' (occupation=20)? 3](#_Toc342652106)

[What is the distribution of ratings for The Rock given by writers? (Tricky :) 3](#_Toc342652107)

[The MovieLens data is obviously normalized and thus the worst possible format for a document-based NoSQL database. Design a new 'schema' in JSON that would be much more effecient for Mongo. 5](#_Toc342652108)

[Hand-craft a small set of documents (5-10) using your new schema which examplify MovieLens data, put them into a new collection, and redesign the above queries so they operate on the new schema. 5](#_Toc342652109)

[Optional: Design a (set of) map-reduce functions that will translate the three MovieLens collections into a single collection with your newly defined schema (denormalization). (Tricky, I haven't been able to do it (yet) :) 5](#_Toc342652110)

[Optional: Use two/more machines to explore the auto-sharding facilities in MongoDB. 5](#_Toc342652111)

[Optional: Use three machines to explore the replication facilities in MongoDB. 6](#_Toc342652112)

[Optional: Read the MovieLens data into a SQL database of your choice and compare performance between SELECTS and Map-Reduce. (Postings on the web generally state that Map Reduce is slow in Mongo.) 6](#_Toc342652113)

# Introduktion

Dette dokument indeholder besvarelsen på opgave H6 i kurset Software Arkitektur i Praksis.

# Find the ID of the film "The Rock" from 1996? (Use find()):

|  |
| --- |
| db.movies.find({ title: "The Rock" },{\_id: 1}) |

No such movie found.

A little searching using:

|  |
| --- |
| db.movies.find({ title: /Rock/ }) |

revelas a “Rock, The (1996)”, and inserting this we get:

|  |
| --- |
| db.movies.find({ title: "Rock, The (1996)" },{\_id: 1}) |

id = 733

# Find how many 5-star ratings "The Rock" has received received? (Use find())

|  |
| --- |
| db.ratings.count({ rating: 5, movie\_id: 733 }) |

= 299

# Calculate a document/cursor describing the distribution of ratings of The Rock? (That is, how many 1, 2, 3, 4, 5 star ratings?) (Use mapReduce())

|  |
| --- |
| var mapFunction1 = function() {  var value = 1;  emit(this.rating, value);  };  var reduceFunction1 = function(key, values) {  return Array.sum(values);  };  db.ratings.mapReduce(  mapFunction1,  reduceFunction1,  { out: "ratings\_distribution", query: { movie\_id: 733 } }  ) |

# Calculate the average rating of The Rock? (Use mapReduce(). You may calculate it on either the original ratings collection or on the output collection from the previous mapReduce.) [I get 3.723]

|  |
| --- |
| var mapFunction1 = function() {  emit(this.movie\_id, this.rating);  };  var reduceFunction1 = function(key, values) {  return values.length; // Array to finalize  return Array.sum(values) / values.length;  };  db.ratings.mapReduce(  mapFunction1,  reduceFunction1,  { out: { inline: 1 }, query: { movie\_id: 733 } }  ) |

= 3.6444281524926687

Unfortunately this is not correct. The error should be found in the reduce function. As there is only one key it is only one

# Count how many movies are classifed as 'Animation' movies?

|  |
| --- |
| db.movies.count( { genres: { $all: [ "Animation" ] } } ) |

= 205

# Generate a collection "drama" containing all titles of movies classified as genre Drama.

|  |
| --- |
| var x = db.movies.find( { genres: { $all: [ "Drama" ] } }, { \_id: 0,title: 1 } )  for (var i = 0; i < x.length(); ++i) {  db.drama\_movies.insert( { movie : x[i] });  }  // Alternative using mapReduce  var mapFunction1 = function() {  emit(this.\_id, this.title);  };  var reduceFunction1 = function(key, values) {  return values;  };  db.movies.mapReduce(  mapFunction1,  reduceFunction1,  { out: "drama\_movies", query: { genres: { $all: [ "Drama" ] } } }  ) |

# How many users have profession 'writer' (occupation=20)?

|  |
| --- |
| db.users.count({ occupation: 20 }) |

= 281

# What is the distribution of ratings for The Rock given by writers? (Tricky :)

|  |
| --- |
| var x = db.users.find({ occupation: 20 },{\_id:1})  // Attempt one - n possible as x may not be used in this way  var mapFunction1 = function() {  emit(this.rating, value);  };  var reduceFunction1 = function(key, values) {  return Array.sum(values);  };  db.ratings.mapReduce(  mapFunction1,  reduceFunction1,  { out: "ratings\_distribution", query: { movie\_id: 733, user\_id: { $in: x } } }  )  // Attempt two - n possible as x may not be used in this way  var mergeUsersAndRatings = function() {  var counts = { };  var values = { };  for (var k = 1; k < 6; ++k) {  values["rating." + k] = 0;  values["count." + k] = 0;  }    var x = db.users.find({ occupation: 20 },{\_id:1})  var y = db.ratings.find( { movie\_id: 733 }, { \_id:0,user\_id:1,rating:1 } )  var n = 0;  for (var i = 0; i < y.length(); ++i) {  var found = 0;  for (var j = 0; j < x.length() && found == 0; ++j) {  if (y[i].user\_id == x[j].\_id) {  found = 1;  var rating = y[i].rating;  values["rating." + rating] += rating;  values["count." + rating] += 1;  ++n;  }  }  }  for (var k = 1; k < 6; ++k) {  values["average." + k] = values["rating." + k] / values["count." + k];  }  return values;  }  // not a very efficient method, but it works  // Attempt three – using mapReduce merge  var mapFunctionUsers = function() {  var data = {  user\_id: this.\_id,  occupation: this.occupation  }  emit(this.\_id,data);  };  var reduceFunctionUsers = function(key, values) {  var result = {  user\_id: 0,  occupation: 0,  ratings: []  };  values.forEach(function(value) {  var field;  if ("rating" in value) {  if (!("ratings" in result)) {  result.ratings = [];  }  result.ratings.push(value.rating);  }  else if ("occupation" in value) {  result.occupation = value.occupation;  }  result.user\_id = value.user\_id;  });  result.user\_id = key;  return result;  };  db.users.mapReduce(  mapFunctionUsers,  reduceFunctionUsers,  { out: { reduce: "user\_rating" } }  )  var mapFunctionRatings = function() {  var data = {  user\_id: this.user\_id,  rating: this.rating  }  emit(this.user\_id,data);  };  db.users.mapReduce(  mapFunctionRatings,  reduceFunctionUsers,  { out: { reduce: "user\_rating2" } }  )  db.users.find().forEach( function(x){db.users2.insert(x)} );  var mapFunctionRatings = function() {  emit(this.user\_id,this.rating);  };  var reduceFunctionRatings = function(key, values) {  var ratings = { };  for (var k = 1; k < values.length; ++k) {  ratings[k] = values[k];  }  return ratings;  };  db.ratings.mapReduce(  mapFunctionRatings,  reduceFunctionRatings,  { out: { merge: "users2" } }  ) |

# The MovieLens data is obviously normalized and thus the worst possible format for a document-based NoSQL database. Design a new 'schema' in JSON that would be much more effecient for Mongo.

I Mongo er der naturligvis ingen skemaer ☺, men for at gøre det sår effektivt så muligt i Mongo skal man have det hele i en collection.

|  |
| --- |
| Ratings: { "\_id", "movie\_id" : 733, "user\_id" : 1, "rating" : 3 }  Movies: { " \_id" : 20, "title" : "Money Train (1995)", "genres" : "Action" }  Users: { "\_id" : 20, "gender" : "M", "age" : 25, "occupation" : 14, "zip\_code" : "55113" }  Combined to:  { "\_id", "movie\_id", "title", "genres", "user\_id", "gender", "age", "occupation", "zip\_code", "rating" } |

# Hand-craft a small set of documents (5-10) using your new schema which examplify MovieLens data, put them into a new collection, and redesign the above queries so they operate on the new schema.

# Optional: Design a (set of) map-reduce functions that will translate the three MovieLens collections into a single collection with your newly defined schema (denormalization). (Tricky, I haven't been able to do it (yet) :)

# Optional: Use two/more machines to explore the auto-sharding facilities in MongoDB.

# Optional: Use three machines to explore the replication facilities in MongoDB.

# Optional: Read the MovieLens data into a SQL database of your choice and compare performance between SELECTS and Map-Reduce. (Postings on the web generally state that Map Reduce is slow in Mongo.)