MongoDB - Assignment 3 - SWEN 432

Zoltan Debre - 300360191

Original repository and progress history: https://github.com/zoltan-nz/mongodb-exercise

Part I

Question 1 - Making your reserves Collection

(32 marks)

To spare you from doing a tedious and repetitive job, Pavle made his reserves MongoDB collection and exported it into the file: `reserves_17.txt`.

You will find the file `reserves_17.txt` on the course Assignments page. Start a single sharded MongoDB deployment and import the file `reserves_17.txt`. (Import command is given in the Appendix.) Call your collection `reserves`. This is a must. (Note: `The reserves_17.txt` may contain some valid documents that are not shown in Assignment 3 Data.)

Soon after exporting the file `reserves_17.txt`, Pavle realized that it contained a number of errors and that it was not complete. In this question, you need to tidy and complete your `reserves` collection.

Importing the database:

```
$ mongoimport --db ass3 --collection reserves  --file ./reserves_17.txt

$ mongo
MongoDB shell version: 2.6.12
connecting to: test
> use ass3
switched to db ass3
> show collections
reserves
system.indexes
>db.reserves.find().length()
14
```

---

a) (2 marks) The name of the `Port Nicholson` marina is misspelled in a number of ways. Use `multi` option of the `db.collection.update()` method to bring it in order.

```
> use ass3
switched to db ass3
> db.reserves.update({ 'marina.name': { $regex: /^port n/im }}, { $set:
{ 'marina.name': 'Port Nicholson' }}, { multi: true })
  WriteResult({ "nMatched" : 7, "nUpserted" : 0, "nModified" : 6 })
```

We can use the following command from MongoDB v3.2:

```
> db.reserves.updateMany('marina.name': { $regex: /^port n/im }}, { $set:
{ 'marina.name': 'Port Nicholson' } })
```

b) (4 marks) In the document `"_id" : ObjectId("54f102de0b54b61a031776ed")`, the field number is misspelled as `numbver`. Rename it.

```
> db.reserves.update({'_id': ObjectId('54f102de0b54b61a031776ed')}, { $rename:
{ 'reserves.boat.numbver': 'reserves.boat.number' } } )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

From MongoDB v3.2

```
> db.reserves.findOneAndUpdate({_id: '54f102de0b54b61a031776ed'},
{ $rename: { 'reserves.boat.numbver': 'reserves.boat.number' } })
```

c) (4 marks) A document for the row

Port Nicholson	**717**	**919**	**2017-03-25**

of the `reserves` table is missing. Insert it.

```
> db.reserves.insert({
    "marina" : { "name" : "Port Nicholson", "location" : "Wellington" },
    "reserves": {
        "boat" : { "name" : "Tarakihi", "number" : 717, "color" : "red",
            "driven_by" : [ "row", "motor" ] },
        "sailor" : { "name" : "Eileen", "sailorId" : 919, "skills" : [ "sail",
            "motor", "swim" ], "address" : "Lower Hutt" },
        "date" : "2017-03-25"
        }
    })
WriteResult({ "nInserted" : 1 })
```

d) (5 marks) We need all boats to be in our database, but the boat `Dolphin`, number `110` from `Port Nicholson` marina had no reserves yet and its data are missing. Make a document containing Dolphin's data and store it in your collection. The document should follow the structure of other documents in the `reserves` collection to the highest possible (and reasonable) extent.

```
> var dolphin = { name: "Dolphin", number: 110, color: "white", driven_by: [] }
> var reserve = { marina: {}, reserves: { boat: dolphin, sailor: {}, date: "" }}
> db.reserves.insert(reserve)
WriteResult({ "nInserted" : 1 })
```

e) (5 marks) We need all sailors to be in our database, but the sailor `Paul` from `Upper Hutt` did not make any reserves yet and his data are missing. Make a document containing Paul's data and store it in your collection. The document should follow the structure of other documents in the `reserves` collection to the highest possible (and reasonable) extent.

```
> var paul = { name: "Paul", sailorId: 110, skills: [ "row", "swim" ],
    address: "Upper Hutt" }
> var reserve = { marina: {}, reserves: { boat: {}, sailor: paul, date: ""}}
> db.reserves.insert(reserve)
```

f) (12 marks) Pavle also realized that he missed to define the following unique constraints:

i. A sailor can make at most one reservation on a given day. (4 marks)

```
> db.reserves.ensureIndex({ "reserves.sailor": 1, "reserves.date": -1 },
    { unique: true} )
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 1,
    "numIndexesAfter" : 2,
    "ok" : 1
}
```

ii. A boat can have at most one reservation on a given day. (4 marks)

```
> db.reserves.ensureIndex({ "reserves.boat": 1, "reserves.date": -1 },
    { unique: true} )
{
    "createdCollectionAutomatically" : false,
    "numIndexesBefore" : 2,
    "numIndexesAfter" : 3,
    "ok" : 1
}
```

iii. Check whether your indexes perform as expected. In your answer, show how did you perform checking, and what messages you received. (4 marks)

```
> var sailor1 = { name: "Charmain", sailorId: 999, skills: [ "row" ],
    address: "Upper Hutt" }
> var sailor2 = { name: "Paul", sailorId: 110, skills: [ "row", "swim" ],
    address: "Upper Hutt"}

> var marina1 = {name: "Evans Bay", location: "Wellington" }
> var marina2 = {name: "Port Nicholson", location: "Wellington"}

> var boat1 = {name: "Night Breeze", number: 818, color: "black",
    driven_by: [ "row" ]}
> var boat2 = {name: "Killer Whale", number: 111, color: "black",
    driven_by: [ "row" ]}

> var date1 = "2017-03-22"
> var date2 = "2017-03-23"

> db.reserves.insert({marina: marina1, reserves: { boat: boat1, sailor: sailor1,
    date: date1}})
WriteResult({ "nInserted" : 1 })

> db.reserves.insert({marina: marina2, reserves: { boat: boat2, sailor: sailor1,
    date: date1}})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 11000,
        "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicate key " +
         "error index: ass3.reserves.$reserves.sailor_1_reserves.date_-1 dup " +
          "key: { : { name: \"Charmain\", sailorId: 999.0, skills: [ \"row\" ], " +
           "address: \"Upper Hutt\" }, : \"2017-03-22\" }"
    }
})

> db.reserves.insert({marina: marina1, reserves: { boat: boat1, sailor: sailor2,
    date: date1}})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 11000,
        "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicate key " +
         "error index: ass3.reserves.$reserves.boat_1_reserves.date_-1  dup " +
          "key: { : { name: \"Night Breeze\", number: 818.0, color: \"black\", " +
           "driven_by: [ \"row\" ] }, : \"2017-03-22\" }"

> db.reserves.insert({marina: marina1, reserves: { boat: boat1, sailor: sailor1,
    date: date2}})
WriteResult({ "nInserted" : 1 })

> db.reserves.insert({marina: marina2, reserves: { boat: boat2, sailor: sailor2,
    date: date2}})
WriteResult({ "nInserted" : 1 })

> db.reserves.insert({marina: marina2, reserves: { boat: boat2, sailor: sailor1,
    date: date2}})
WriteResult({
    "nInserted" : 0,
    "writeError" : {
        "code" : 11000,
        "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicate key " +
         "error index: ass3.reserves.$reserves.sailor_1_reserves.date_-1 " +
          "dup key: { : { name: \"Charmain\", sailorId: 999.0, skills: " +
           "[ \"row\" ], address: \"Upper Hutt\" }, : \"2017-03-23\" }"
    }
})
```

```
> db.reserves.getIndexes()
[
    {
        "v" : 1,
        "key" : {
            "_id" : 1
        },
        "name" : "_id_",
        "ns" : "ass3.reserves"
    },
    {
        "v" : 1,
        "unique" : true,
        "key" : {
            "reserves.sailor" : 1,
            "reserves.date" : -1
        },
        "name" : "reserves.sailor_1_reserves.date_-1",
        "ns" : "ass3.reserves"
    },
    {
        "v" : 1,
        "unique" : true,
        "key" : {
            "reserves.boat" : 1,
            "reserves.date" : -1
        },
        "name" : "reserves.boat_1_reserves.date_-1",
        "ns" : "ass3.reserves"
    }
]
```

Question 2 - Simple Queries

(26 marks)

Find all script together in `q2.js`.

a) (2mark) Find the number of all documents in your `reserves` collection.

using `length()`

```
> use ass3
switched to db ass3
> db.reserves.find().length()
20
```

using `count()`

```
> db.reserves.find().count()
20
```

b) (4 marks) Find the number of all documents in your `reserves` collection containing reserves made in `Port Nicholson` marina.

using `length()`

```
> db.reserves.find({ "marina.name": "Port Nicholson" }).length()
10
```

using `count()`

```
> db.reserves.find({ "marina.name": "Port Nicholson" }).count()
10
```

c) (4 mark) Find unique sailor names.

```
> db.reserves.distinct( "reserves.sailor.name" )
  [
    "James",
    "Peter",
    "Milan",
    "Eileen",
    "Charmain",
    "Gwendolynn",
    "Paul"
  ]
```

d) (5 marks) Find marina names, boat names, and sailor names having a reservation on `2017-03-16`.

```
> db.reserves.find({ "reserves.date": "2017-03-16" }, { _id: 0,  "marina.name":
    1, "reserves.boat.name": 1, "reserves.sailor.name": 1})

  { "marina" : { "name" : "Sea View" }, "reserves" : { "boat" : { "name" :
    "Flying Dutch" }, "sailor" : { "name" : "Peter" } } }
  { "marina" : { "name" : "Port Nicholson" }, "reserves" : { "boat" : { "name" :
    "Mermaid" }, "sailor" : { "name" : "Milan" } } }
```

e) (5 marks) Find sailors having `swim` skill. Display just sailor names.

```
> db.reserves.distinct( "reserves.sailor.name", { "reserves.sailor.skills":
    "swim" } )

  [ "Eileen", "Paul" ]
```

f) (6 marks) Find sailors having exactly `row`, `sail`, and `motor` skills (no more and no less, but in an arbitrary order). Display just sailor names.

```
> db.reserves.distinct("reserves.sailor.name", { "reserves.sailor.skills":
    { $size: 3, $all: ["motor", "row", "sail"] } })

[ "Peter" ]

> db.reserves.distinct("reserves.sailor.name", { "reserves.sailor.skills":
    { $size: 3, $all: ["row", "sail", "motor"] } })

[ "Peter" ]
```

Question 3 - Time Table Collection

(10 marks)

Assume Wellington Tranz Metro has acquired an iPhone application that works as a data recorder for railway vehicles (`cars`, `engines`). The application uses mobile data connections to send information to servers in real time. The information includes measurements such as the `location` and `speed` of the `vehicle`. They have selected MongoDB (instead of Cassandra, as in Assignment 1) as the CDBMS to use for this project. They invited you, as a respected database specialist to design an embedded collection of the database.

a) (6 marks) Assume your document will contain the following entity types:

- `driver`,
- `vehicle`,

- `timeTable`, and
- `dataPoint`.

Design a MongoDB document by representing relationships between entity types by **embedding**. Make relationships between entity types clearly visible in your design. Use iPhone database sample data of Assignment 1 to populate your document. Show your design in your answer.

We are collection data point sequences from iPhone application. Each data point sequence will be inserted in our collection. Our collection will be called `datapoints` and we insert it with `db.datapoints.insert(sequence1)`. Each inserted document contains details about connected entities: `driver` (with `name`, `currentPosition`, `mobile`, etc.), `vehicle` (with `vehicleId`, `status`, etc.), `timeTable` (with `lineName`, `services` array, in a `service` we have an embeded doc which contains details about a `station`), `date` and the actual `dataPoint`.

Because we embed more level of data, much cleaner if we use variables to manage our sample data. The sample is for demo only and not fully represent a real situation, however it shows the design of the collection.

```
// This script can be run with load('q3.js')
> var stationWellington = {
  name: 'Wellington', longitude: 174.7762, latitude: -41.2865
}

> var stationPetone = {
  name: 'Petone', longitude: 174.8851, latitude: -41.227
}

> var service1 = {
  serviceNo: 1, timeTable: [
    {station: stationWellington, time: '0605', distance: 0},
    {station: stationPetone, time: '0617', distance: 8.3}
  ]
}

> var service11 = {
  serviceNo: 11, timeTable: [
    {station: stationWellington, time: '1935', distance: 0},
    {station: stationPetone, time: '1947', distance: 8.3}
  ]
}

> var huttValeyLineTimeTable =
  {
    lineName: 'Hutt Valey Line (north bound)',
    services: [service1, service11]
  }

> var driver1 = {
  name: 'milan', currentPosition: 'Upper Hutt', mobile: '211111', password: 'mm77',
}

> var vehicle1 = {
  vehicleId: 'FA1122', status: 'Upper Hutt', type: 'Matangi'
}

> var firstDataPoint =
  {sequence: '0610', position: {latitude: 174.77, longitude: -41.2262}, speed: 29.1}

> var secondDataPoint =
  {sequence: '0615', position: {latitude: 175, longitude: -41.2012}, speed: 70.1};

> var sequence1 = {
  driver: driver1,
  vehicle: vehicle1,
  timeTable: huttValeyLineTimeTable,
  date: '2017-03-25',
  dataPoint: firstDataPoint
}

> var sequence2 = {
  driver: driver1,
  vehicle: vehicle1,
  timeTable: huttValeyLineTimeTable,
  date: '2017-03-25',
  dataPoint: secondDataPoint
}

> db.datapoints.insert(sequence1);
WriteResult({ "nInserted" : 1 })
> db.datapoints.insert(sequence2);
WriteResult({ "nInserted" : 1 })

> db.datapoints.find();
{ "_id" : ObjectId("5916a9c9e2029d7a9245d7a3"), "driver" : { "name" : "milan", "curr
"mobile" : "211111", "password" : "mm77", "skills" : [ "Matangi" ] }, "vehicle" : {
"Upper Hutt", "type" : "Matangi" }, "timeTable" : { "lineName" : "Hutt Valey Line (r
[ { "serviceNo" : 1, "timeTable" : [ { "station" : { "name" : "Wellington", "longitu
-41.2865 }, "time" : "0605", "distance" : 0 }, { "station" : { "name" : "Petone", "l
-41.227 }, "time" : "0617", "distance" : 8.3 } ] }, { "serviceNo" : 11, "timeTable"
"Wellington", "longitude" : 174.7762, "latitude" : -41.2865 }, "time" : "1935", "dis
{ "name" : "Petone", "longitude" : 174.8851, "latitude" : -41.227 }, "time" : "1947"
"date" : "2017-03-25", "dataPoint" : { "sequence" : "0610", "position" : { "latitude
-41.2262 }, "speed" : 29.1 } }
```

```
{ "_id" : ObjectId("5916a9cbe2029d7a9245d7a4"), "driver" : { "name" : "milan", "curr
"mobile" : "211111", "password" : "mm77", "skills" : [ "Matangi" ] }, "vehicle" : {
"Upper Hutt", "type" : "Matangi" }, "timeTable" : { "lineName" : "Hutt Valey Line (r
[ { "serviceNo" : 1, "timeTable" : [ { "station" : { "name" : "Wellington", "longitu
-41.2865 }, "time" : "0605", "distance" : 0 }, { "station" : { "name" : "Petone", "l
-41.227 }, "time" : "0617", "distance" : 8.3 } ] }, { "serviceNo" : 11, "timeTable"
"Wellington", "longitude" : 174.7762, "latitude" : -41.2865 }, "time" : "1935", "dis
{ "name" : "Petone", "longitude" : 174.8851, "latitude" : -41.227 }, "time" : "1947'
"date" : "2017-03-25", "dataPoint" : { "sequence" : "0615", "position" : { "latitude
"speed" : 70.1 } }
```

b) (4 marks) How many instances of each entity type may contain your document maximally (make a best guess if you can't give an exact number)?

One document contains: `driver` , `vehicle` , `timeTable` , `service` , `station` , `dataPoint` It is maximum 6 type of entity.

Using our sample database, one inserted document will contain maximum the following number of instances:

- `driver` : 1,
- `vehicle` : 1,
- `timeTable` : 1,
- `service` : 2 (in Assignment 1, all time table have maximum 2 active services),
- `station` : 8 (in Assignment 1, we have data about 8 stations, and the longest timeTable has 8 stops),
- `dataPoint` : not limited, because we collect this continuously.

Part II

Pavle implemented the referencing version of the `Boat Hire` database as four collections and exported them into files:

- `marina_17.txt` ,
- `sailor_17.txt` ,
- `boat_17.txt` ,
- `res_ref_17.tx` t.

You will find these files at the course Assignments page. Start a single sharded MongoDB deployment and import the files. You may do import into the same database as for `reserves` collection. Call the new collections: `marina` , `sailor` , `boat` , and `res_ref` . So far, Pavle did not find any errors in the implementation. Use these collections to answer the following questions.

Importing collections:

```
$ mongoimport --db ass3 --collection marina  --file ./marina_17.txt
coonnected to: 127.0.0.1
imported 3 objects
$ mongoimport --db ass3 --collection sailor --file ./sailor_17.txt
imported 7 objects
$ mongoimport --db ass3 --collection boat --file ./boat_17.txt
check 9 13
imported 13 object
$ mongoimport --db ass3 --collection res_ref --file ./res_ref_17.txt
check 9 15
imported 15 objects
```

```
$ mongo
> use ass3
switched to db ass3
> show collections
  boat
  datapoints
  marina
  res_ref
  reserves
  sailor
  system.indexes
```

Question 4. - Simple Queries

(8 marks)

Find all script together in `q4.js`.

a) (4 marks) Find all unique sailors.

Let's suppose we have a constrain index in our `sailor` database which refuse duplication, so in this case a simple `find` can list all the unique sailor.

```
> use ass3
> db.sailor.find()
```

We can use `distinct` to get unique values, for example unique `names`:

```
> db.sailor.distinct('name')
[
    "James",
    "Peter",
    "Milan",
    "Eileen",
    "Paul",
    "Charmain",
    "Gwendolynn"
]
```

Of course we could have two sailors with the same name, but they must have different `sailorId`. Let's say, there was a mistake, and there was not implemented an index constrain, so the same data (with the same `sailorId`) was inserted in our collection. In this case we can use aggregation and groups. The following query will show only the first instance if there is two records with the same data (`sailorId`, `name`, `skills`, `address`)

```
>  db.sailor.aggregate({$group: { '_id': '$sailorId', sailor: { $first: { 'name': '$
    'address': '$address' } } }})
{ "_id" : 777, "sailor" : { "name" : "Gwendolynn", "skills" : [ "row", "sail", "moto
    "Masterton" } }
{ "_id" : 110, "sailor" : { "name" : "Paul", "skills" : [ "row", "swim" ], "address'
{ "_id" : 919, "sailor" : { "name" : "Eileen", "skills" : [ "sail", "motor", "swim"
{ "_id" : 111, "sailor" : { "name" : "Peter", "skills" : [ "row", "sail", "motor" ],
{ "_id" : 999, "sailor" : { "name" : "Charmain", "skills" : [ "row" ], "address" : '
{ "_id" : 818, "sailor" : { "name" : "Milan", "skills" : [ "row", "sail", "motor", '
    "Wellington" } }
{ "_id" : 707, "sailor" : { "name" : "James", "skills" : [ "row", "sail", "motor", '
    "Wellington" } }
```

b) (4 marks) Find sailors having exactly `row`, `sail`, and `motor` skills (no more and no less). Display just sailor names.

```
> db.sailor.distinct("name", { "skills": { $size: 3, $all: ["motor", "row", "sail"]
[ "Peter" ]
```

Question 5 - Complex Queries

(24 marks)

a) (8 marks) Find `marina names`, `boat names`, and `sailor names` having a reservation on "`2017-03-16`".

```
// Find this script in `q5a.js` and run in mongo shell with load('q5a.js')

// Don't forget to switch to the relevant database in mongo shell, for example: `use

var dateFilter = '2017-03-16';

var reservationsCursor = db.res_ref.find(
  { 'reserves.date': dateFilter },
  { '_id': 0, marina: 1, 'reserves.boat': 1, 'reserves.sailor': 1 }
);

var reservations = reservationsCursor.map(function(reservation) {
  var marinaName = reservation.marina;
  var boatNumber = reservation.reserves.boat;
  var sailorId = reservation.reserves.sailor;

  // There are boats with the same number but in different marina, for this reason o
  // accurate if we specify the marina name also.
  var boat = db.boat.findOne({ marina: marinaName, number: boatNumber }, { '_id': 0,
  // With this guard, our script safe if there isn't any boat object. Try it out wit
  var boatName = boat ? boat.name : '';

  var sailor = db.sailor.findOne({ sailorId: sailorId }, { '_id': 0, name: 1 });
  var sailorName = sailor ? sailor.name : '';

  return { marinaName: marinaName, boatName: boatName, sailorName: sailorName };
});

print(tojson(reservations));
```

- Run the above script with `load('q5a.js')`
- Result:

```
> use ass3
> load('q5a.js')
[
    {
        "marinaName" : "Sea View",
        "boatName" : "Flying Dutch",
        "sailorName" : "Peter"
    },
    {
        "marinaName" : "Port Nicholson",
        "boatName" : "Mermaid",
        "sailorName" : "Milan"
    }
]
true
```

The above script merged together in one command and it is the answer for b) See below.

b) (16 marks) Find `marina names`, `boat names`, and `sailor names` having a reservation on "`2017-03-16`" using not more than two commands in mongo shell.

```
// q5b.js
print(tojson(
  db.res_ref.find(
    { 'reserves.date': '2017-03-16' },
    { '_id': 0, marina: 1, 'reserves.boat': 1, 'reserves.sailor': 1 }
  ).map(function(reservation) {
    var marinaName = reservation.marina;
    var boatNumber = reservation.reserves.boat;
    var sailorId = reservation.reserves.sailor;

    var boat = db.boat.findOne({ marina: marinaName, number: boatNumber }, { '_id':
    var boatName = boat ? boat.name : '';

    var sailor = db.sailor.findOne({ sailorId: sailorId }, { '_id': 0, name: 1 });
    var sailorName = sailor ? sailor.name : '';

    return { marinaName: marinaName, boatName: boatName, sailorName: sailorName };
  })
));
```