MongoDB - Assignment 4 - SWEN 432

Zoltan Debre - 300360191

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

Part I

To answer questions in this part of the assignment, use the collection reserves you imported from the file ass4_reserves.json.

Importing…

```
$ mongoimport --db ass4 --collection reserves
    --file ./assignment-4/ass4_reserves_17.json
Connected to: 127.0.0.1
check 9 22
imported 22 objects
```

```
$ mongo
MongoDB shell version: 2.6.12
connecting to: test
> use ass4
switched to db ass4
> show collections
reserves
system.indexes
> db.reserves.count()
22
```

Question 1

(10 marks)

Retrieve all unique sailors. In your answer, sailor documents should have the structure of a simple (non embedded) document, like:

```
{
  _id: <'reserves.sailor.sailorId'>,
  name: <'reserves.sailor.name'>,
  skills: <'reserves.sailor.skills'>,
  address: <'reserves.sailor.address'>
}
```

e.g.

```
{"_id": 110, "name": "Paul", "skills": ["row"], "address": "Upper Hutt"}
```

Answer:

```
> db.reserves.aggregate([
  { $match: { 'reserves.sailor.sailorId': { $exists: true } } },
  {
    $group: {
      _id: '$reserves.sailor.sailorId',
      name: { $first: '$reserves.sailor.name' },
      skills: { $first: '$reserves.sailor.skills' },
      address: { $first: '$reserves.sailor.address' }
    }
  }
]);

{ "_id" : 110, "name" : "Paul", "skills" : [ "row", "swim" ], "address" :
  "Upper Hutt" }
{ "_id" : 777, "name" : "Alva", "skills" : [ "row", "sail", "motor", "dance" ],
  "address" : "Masterton" }
{ "_id" : 919, "name" : "Eileen", "skills" : [ "sail", "motor", "swim" ],
  "address" : "Lower Hutt" }
{ "_id" : 111, "name" : "Peter", "skills" : [ "row", "sail", "motor" ],
  "address" : "Upper Hutt" }
{ "_id" : 999, "name" : "Charmain", "skills" : [ "row" ], "address" :
  "Upper Hutt" }
{ "_id" : 818, "name" : "Milan", "skills" : [ "row", "sail", "motor",
  "first aid" ], "address" : "Wellington" }
{ "_id" : 707, "name" : "James", "skills" : [ "row", "sail", "motor", "fish" ],
  "address" : "Wellington" }
```

Question 2

(14 marks)

Find the `sailor` who made the *maximum number* of reservations. In your answer, sailor documents should have the structure of a simple (non embedded) document, containing fields: `sailorId` , `name` , `address` , and `no_of_reserves` .

Answer:

Assuming a reservation is valid if it has a valid `reserves.date` and a `sailor` .

```
> db.reserves.aggregate([
  { $match: { $and: [{ 'reserves.date': { $exists: true } },
    { 'reserves.sailor': { $exists: true } }] } },
  {
    $group: {
      _id: '$reserves.sailor.sailorId',
      name: { $first: '$reserves.sailor.name' },
      address: { $first: '$reserves.sailor.address' },
      no_of_reserves: { $sum: 1 }
    }
  }, {
    $project: {
      _id: 0,
      sailorId: '$_id',
      name: 1,
      address: 1,
      no_of_reserves: 1
    }
  },
  { $sort: { no_of_reserves: -1 } },
  { $limit: 1 }
]);

{ "name" : "Milan", "address" : "Wellington", "no_of_reserves" : 6,
  "sailorId" : 818 }
```

Question 3

(6 marks)

Find the total number of `reserves` made by `sailors` . In your answer, the output document should contain just one field with the name `total_reserves` .

Answer:

Assuming a reservation is valid if it has a valid `reserves.date` and a `sailor` . (However, we don't have any document in our sample data set which has date without sailor, so we can less strict in our later tasks. But keeping this assumption here help to show the usage of `$and` in our aggregation.)

```
> db.reserves.aggregate([
  { $match: { $and: [{ 'reserves.date': { $exists: true } },
      { 'reserves.sailor': { $exists: true } }] } },
  { $group: { _id: null, total_reserves: { $sum: 1 } } },
  { $project: { _id: false, total_reserves: true } }
]);

{ "total_reserves" : 18 }
```

Question 4

(22 marks)

Find the average number of reserves made by all sailors. If you develop a statement that contains a multistage aggregate() method that produces a correct result, you get 22 marks. If you develop a multi step procedure using manual interventions and get the correct result, you get 14 marks.

Hint: The result produced by your (single) pipeline aggregation statement may be incorrect. Perform a manual checking. If you realize your statement produced an incorrect result, explain why it did and develop one that produces a correct result.

Answer:

We know from the result of Question 3, that we have `18` valid reservations. We know from the result of Question 1, that we have `7` unique sailor. The average number of reserves made by all sailors should be `18 / 7 = 2.5714`

If we filter our database for `reserves.date`, we will miss one of the sailor who does not have any reservation, so the result will be `3` which is **INCORRECT**:

```
db.reserves.aggregate([
  { $match: { 'reserves.date': { $exists: true } } },
  { $group: { _id: '$reserves.sailor.sailorId',
      no_of_reserves_by_sailor: { $sum: 1 } } },
  { $group: { _id: null, average_number_of_reserves_by_all_sailors:
      { $avg: '$no_of_reserves_by_sailor' } } },
  { $project: { _id: false, average_number_of_reserves_by_all_sailors: true } }
]);

{ "average_number_of_reserves_by_all_sailors" : 3 }
```

After experimenting with a few hours, I realized, the best option to solve this task to collect dates and sailors in two sets, checking the size of these arrays and using `divide` to calculate the average number.

Important to note, that we should not use any filter (`$match`), because we want to use the whole dataset to calculate the right number.

In the first step we use `$push` to add each date to our array. `Push` keeps duplications, so the size of the array will represent the valid number of reservations. (Assuming reservation is valid if `reserves.date` exists.) Using `$addToSet` for sailors helps to have a clean list of unique sailor ids (without duplication). The size of this set shows the exact number of sailors and we add sailors with zero reservations as well.

In the second step, we just use `$size` to get the length of our sets.

In the third step we calculate the average number with `$divide` :

```
> db.reserves.aggregate([
  {
    $group: {
      _id: null,
      each_reserves_date: { $push: '$reserves.date' },
      unique_sailor_ids: { $addToSet: '$reserves.sailor.sailorId' },
      counter: { $sum: 1 }
    }
  },
  {
    $project: {
      _id: 0,
      no_of_reserves: { $size: '$each_reserves_date' },
      no_of_sailors: { $size: '$unique_sailor_ids' }
    }
  },
  {
    $project: {
      average_number_of_reserves_by_all_sailors:
        { $divide: ['$no_of_reserves', '$no_of_sailors'] }
    }
  },
]);

{ "average_number_of_reserves_by_all_sailors" : 2.5714285714285716 }
```

Question 5

(10 + 15 marks)

The sailor Paul from Upper Hutt made a reservation for the boat Mermaid in the Port Nicholson marina. When a supervisor checked accepted reserves, he realized that according to Paul’s skills, he does not qualify to drive Mermaid. So, people from the Port Nicholson marina asked you, as a respected MongoDB expert, to design a procedure that will return all boats that a given sailor is qualified to drive. What will be your solution to the problem?

Note: A sailor is qualified to drive a boat only if the boat's `driven_by` array is not empty (null) and is a subset of the sailor's `skills` array.

The following script prints out a detailed list about boats and driver. (This is the same script what you can find in `assignment-4.js` file.)

```
> var sailorName = 'Paul';
> var sailorSkills = db.reserves.distinct('reserves.sailor.skills',
    { 'reserves.sailor.name': sailorName });
> print('Sailor: ' + sailorName);
Sailor: Paul
> print('Skills: ' + sailorSkills);
Skills: row,swim

> var q5detailed = db.reserves.aggregate([
  { $match: { 'reserves.boat.driven_by': { $exists: true, $ne: [] } } },
  {
    $group: {
      _id: '$reserves.boat.number',
      name: { $first: '$reserves.boat.name' },
      driven_by: { $first: '$reserves.boat.driven_by' }
    }
  },
  {
    $project: {
      _id: false,
      boat_number: '$_id',
      name: true,
      driven_by: true,
      sailor_can_drive: { $setIsSubset: ['$driven_by', sailorSkills] }
    }
  }
]);> q5detailed.shellPrint();

{ "name" : "Penguin", "driven_by" : [ "row" ], "boat_number" : 131,
  "sailor_can_drive" : true }
{ "name" : "Night Breeze", "driven_by" : [ "row" ], "boat_number" : 818,
  "sailor_can_drive" : true }
{ "name" : "Mermaid", "driven_by" : [ "sail", "motor" ], "boat_number" : 919,
  "sailor_can_drive" : false }
{ "name" : "Sea Gull", "driven_by" : [ "row" ], "boat_number" : 137,
  "sailor_can_drive" : true }
{ "name" : "Tarakihi", "driven_by" : [ "row", "motor" ], "boat_number" : 717,
  "sailor_can_drive" : false }
{ "name" : "Red Cod", "driven_by" : [ "sail", "motor" ], "boat_number" : 616,
  "sailor_can_drive" : false }
{ "name" : "Dolphin", "driven_by" : [ "sail", "motor" ], "boat_number" : 110,
  "sailor_can_drive" : false }
{ "name" : "Blue Shark", "driven_by" : [ "motor" ], "boat_number" : 515,
  "sailor_can_drive" : false }
{ "name" : "Killer Whale", "driven_by" : [ "row" ], "boat_number" : 111,
  "sailor_can_drive" : true }
{ "name" : "Flying Dutch", "driven_by" : [ "sail" ], "boat_number" : 313,
  "sailor_can_drive" : false }
```

The following script generates a simple array with suitable boat names for a specified sailor. We use our `sailorSkills` variable from above:

```
> var q5onlyBoatNames = db.reserves.aggregate([
  { $match: { 'reserves.boat.driven_by': { $exists: true, $ne: [] } } },
  {
    $group: {
      _id: '$reserves.boat.number',
      name: { $first: '$reserves.boat.name' },
      driven_by: { $first: '$reserves.boat.driven_by' }
    }
  },
  {
    $group: {
      _id: null,
      boats: { $addToSet: { $cond: [{ $setIsSubset: ['$driven_by',
        sailorSkills] }, '$name', 0] } }
    }
  },
  { $unwind: '$boats' },
  { $match: { boats: { $type: 2 } } },
  { $group: { _id: null, boatNames: { $addToSet: '$boats' } } },
  { $project: { _id: false, boatNames: true }}
]);
> q5onlyBoatNames.shellPrint();

{ "boatNames" : [ "Penguin", "Night Breeze", "Sea Gull", "Killer Whale" ] }
```

Bonus: Use aggregate() method and JavaScripts to find `sailors` who made more than `average reserves`. You will get 15 bonus marks if you get a correct result, but do not insert manually constants into `aggregate()` method stages to answer the question.

Answer:

Assuming the "constants" means a number and not a JavaScript variable which is calculated in a previous step. If a calculated JavaScript variable allowed, the following could be a solution. (Please find it in the attached `bonus.js` file.)

I use the combination of our scripts from previous questions.

```
var averageCursor = db.reserves.aggregate([
  {
    $group: {
      _id: null,
      each_reserves_date: { $push: '$reserves.date' },
      unique_sailor_ids: { $addToSet: '$reserves.sailor.sailorId' },
      counter: { $sum: 1 }
    }
  },
  {
    $project: {
      _id: 0,
      no_of_reserves: { $size: '$each_reserves_date' },
      no_of_sailors: { $size: '$unique_sailor_ids' }
    }
  },
  {
    $project: {
      average: { $divide: ['$no_of_reserves', '$no_of_sailors'] }
    }
  },
]);

var average = averageCursor.next().average;

var sailorsAboveAverage = db.reserves.aggregate([
  { $match: { $and: [{ 'reserves.date': { $exists: true } },
      { 'reserves.sailor': { $exists: true } }] } },
  {
    $group: {
      _id: '$reserves.sailor.sailorId',
      name: { $first: '$reserves.sailor.name' },
      address: { $first: '$reserves.sailor.address' },
      no_of_reserves: { $sum: 1 }
    }
  }, {
    $project: {
      _id: 0,
      sailorId: '$_id',
      name: 1,
      address: 1,
      no_of_reserves: 1
    }
  },
  { $sort: { no_of_reserves: -1 } },
  { $match: { no_of_reserves: {$gt: average } }}
]);
sailorsAboveAverage.shellPrint();

$ mongo localhost/ass4 ./bonus.js
MongoDB shell version: 2.6.12
connecting to: localhost/ass4
{ "name" : "Milan", "address" : "Wellington", "no_of_reserves" : 6,
  "sailorId" : 818 }
{ "name" : "Peter", "address" : "Upper Hutt", "no_of_reserves" : 3,
  "sailorId" : 111 }
{ "name" : "James", "address" : "Wellington", "no_of_reserves" : 3,
  "sailorId" : 707 }
```

Part II

Question 6. Sharding

Read carefully all subquestions of the question. You will use the same MongoDB deployment to answer several subquestions. Use the sha-mongo script to produce deployments with 2 shards, 5 shards, and 10 shards. Populate each of them by the user collection using the test parameter to the sha-mongo script.

Answers:

```
$ sha-mongo init 2
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("59254cd5a19df78a0cc2a406")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
```

```
$ sha-mongo init 5
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("59254cd5a19df78a0cc2a406")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
    {  "_id" : "shard0002",  "host" : "127.0.0.1:27022" }
    {  "_id" : "shard0003",  "host" : "127.0.0.1:27023" }
    {  "_id" : "shard0004",  "host" : "127.0.0.1:27024" }


$ sha-mongo init 10
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("59254cd5a19df78a0cc2a406")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
    {  "_id" : "shard0002",  "host" : "127.0.0.1:27022" }
    {  "_id" : "shard0003",  "host" : "127.0.0.1:27023" }
    {  "_id" : "shard0004",  "host" : "127.0.0.1:27024" }
    {  "_id" : "shard0005",  "host" : "127.0.0.1:27025" }
    {  "_id" : "shard0006",  "host" : "127.0.0.1:27026" }
    {  "_id" : "shard0007",  "host" : "127.0.0.1:27027" }
    {  "_id" : "shard0008",  "host" : "127.0.0.1:27028" }
    {  "_id" : "shard0009",  "host" : "127.0.0.1:27029" }
```

```
$ sha-mongo test
$ sha-mongo connect
```

a) (2 marks) Which partitioning method has been used for sharding the user collection?

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("5925525de03999ba48cc0037")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
    {  "_id" : "shard0002",  "host" : "127.0.0.1:27022" }
    {  "_id" : "shard0003",  "host" : "127.0.0.1:27023" }
    {  "_id" : "shard0004",  "host" : "127.0.0.1:27024" }
    {  "_id" : "shard0005",  "host" : "127.0.0.1:27025" }
    {  "_id" : "shard0006",  "host" : "127.0.0.1:27026" }
    {  "_id" : "shard0007",  "host" : "127.0.0.1:27027" }
    {  "_id" : "shard0008",  "host" : "127.0.0.1:27028" }
    {  "_id" : "shard0009",  "host" : "127.0.0.1:27029" }
  databases:
    {  "_id" : "test",  "partitioned" : true,  "primary" : "shard0000" }
```

We can see in `sha-mongo` script that `sha-mongo test` will run the `load_shard.js` script.

On the top of this script, we can see the following:

```
db = db.getSiblingDB('mydb');
sh.enableSharding("mydb");
db.user.ensureIndex({"user_id":1});
sh.shardCollection("mydb.user",{"user_id":1});
```

We use `Ranged Sharding` here, because we declare the shard key with `{ "user_id": 1}`. (If we would use `Hashed Sharding`, the shard key declaration would look like this: `{ "user_id": "hashed" }`.)

Using `sh.status()` in `mongos` shell can show also what is our shard key.

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("59256b655c86719e0dbb2455")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
    {  "_id" : "shard0002",  "host" : "127.0.0.1:27022" }
    {  "_id" : "shard0003",  "host" : "127.0.0.1:27023" }
    {  "_id" : "shard0004",  "host" : "127.0.0.1:27024" }
    {  "_id" : "shard0005",  "host" : "127.0.0.1:27025" }
  databases:
    {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
    {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard0005" }
          mydb.user
              shard key: { "user_id" : 1 }
              chunks:
                  shard0000   1
                  shard0004   1
                  shard0005   1
                  shard0001   1
                  shard0002   1
                  shard0003   1
              { "user_id" : { "$minKey" : 1 } } -->> { "user_id" : 38339 } on :
                  shard0000 Timestamp(2, 0)
              { "user_id" : 38339 } -->> { "user_id" : 40680 } on : shard0004
                  Timestamp(6, 0)
              { "user_id" : 40680 } -->> { "user_id" : 70999 } on : shard0005
                  Timestamp(6, 1)
              { "user_id" : 70999 } -->> { "user_id" : 80999 } on : shard0001
                  Timestamp(4, 1)
              { "user_id" : 80999 } -->> { "user_id" : 90999 } on : shard0002
                  Timestamp(5, 1)
              { "user_id" : 90999 } -->> { "user_id" : { "$maxKey" : 1 } } on
                  : shard0003 Timestamp(5, 0)
```

We can see here: `mydb.user shard key: { "user_id" : 1 }` This key is for `Ranged Sharding`.

b) 1. How many chunks (roughly) has each shard in the case of: 2, 5, and 10 shards? (2 mark)

In the case of 2: Each shard has `6` chunks. (See below.)

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("59256e0650560bc4870b8480")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
  databases:
    {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
    {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard0000" }
          mydb.user
              shard key: { "user_id" : 1 }
              chunks:
                  shard0001   6
                  shard0000   6
              { "user_id" : { "$minKey" : 1 } } -->> { "user_id" : 0 } on :
                  shard0001 Timestamp(12, 0)
              { "user_id" : 0 } -->> { "user_id" : 5999 } on : shard0000
                  Timestamp(12, 1)
              { "user_id" : 5999 } -->> { "user_id" : 15999 } on : shard0001
                  Timestamp(11, 1)
              { "user_id" : 15999 } -->> { "user_id" : 25999 } on : shard0000
                  Timestamp(3, 2)
              { "user_id" : 25999 } -->> { "user_id" : 35999 } on : shard0001
                  Timestamp(4, 2)
              { "user_id" : 35999 } -->> { "user_id" : 45999 } on : shard0000
                  Timestamp(5, 2)
              { "user_id" : 45999 } -->> { "user_id" : 55999 } on : shard0001
                  Timestamp(6, 2)
              { "user_id" : 55999 } -->> { "user_id" : 65999 } on : shard0000
                  Timestamp(7, 2)
              { "user_id" : 65999 } -->> { "user_id" : 75999 } on : shard0001
                  Timestamp(8, 2)
              { "user_id" : 75999 } -->> { "user_id" : 85999 } on : shard0000
                  Timestamp(9, 2)
              { "user_id" : 85999 } -->> { "user_id" : 95999 } on : shard0001
                  Timestamp(10, 2)
              { "user_id" : 95999 } -->> { "user_id" : { "$maxKey" : 1 } } on :
                  shard0000 Timestamp(11, 0)
```

In case of 5: Each shard has `2` chunks. (See below.)

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("59256ecb5fa78da8cab1f3d8")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
    {  "_id" : "shard0002",  "host" : "127.0.0.1:27022" }
    {  "_id" : "shard0003",  "host" : "127.0.0.1:27023" }
    {  "_id" : "shard0004",  "host" : "127.0.0.1:27024" }
  databases:
    {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
    {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard0000" }
        mydb.user
            shard key: { "user_id" : 1 }
            chunks:
                shard0002   2
                shard0000   2
                shard0001   2
                shard0003   2
                shard0004   2
            { "user_id" : { "$minKey" : 1 } } -->> { "user_id" : 0 } on :
                shard0002 Timestamp(9, 1)
            { "user_id" : 0 } -->> { "user_id" : 5999 } on : shard0000
                Timestamp(7, 1)
            { "user_id" : 5999 } -->> { "user_id" : 34999 } on : shard0001
                Timestamp(8, 1)
            { "user_id" : 34999 } -->> { "user_id" : 44999 } on : shard0003
                Timestamp(10, 1)
            { "user_id" : 44999 } -->> { "user_id" : 54999 } on : shard0004
                Timestamp(6, 1)
            { "user_id" : 54999 } -->> { "user_id" : 64999 } on : shard0000
                Timestamp(6, 2)
            { "user_id" : 64999 } -->> { "user_id" : 74999 } on : shard0001
                Timestamp(7, 2)
            { "user_id" : 74999 } -->> { "user_id" : 84999 } on : shard0002
                Timestamp(8, 2)
            { "user_id" : 84999 } -->> { "user_id" : 94999 } on : shard0003
                Timestamp(9, 2)
            { "user_id" : 94999 } -->> { "user_id" : { "$maxKey" : 1 } } on :
                shard0004 Timestamp(10, 0)
```

In case of 10: The first two shards have 2 chunks and the rest of them has only 1 chunk. (See below.)

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("59256f21e93b7c9690eb60b4")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
    {  "_id" : "shard0002",  "host" : "127.0.0.1:27022" }
    {  "_id" : "shard0003",  "host" : "127.0.0.1:27023" }
    {  "_id" : "shard0004",  "host" : "127.0.0.1:27024" }
    {  "_id" : "shard0005",  "host" : "127.0.0.1:27025" }
    {  "_id" : "shard0006",  "host" : "127.0.0.1:27026" }
    {  "_id" : "shard0007",  "host" : "127.0.0.1:27027" }
    {  "_id" : "shard0008",  "host" : "127.0.0.1:27028" }
    {  "_id" : "shard0009",  "host" : "127.0.0.1:27029" }
  databases:
    {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
    {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard0000" }
        mydb.user
            shard key: { "user_id" : 1 }
            chunks:
                shard0000   2
                shard0001   2
                shard0002   1
                shard0003   1
                shard0004   1
                shard0005   1
                shard0006   1
                shard0007   1
                shard0008   1
                shard0009   1
            { "user_id" : { "$minKey" : 1 } } -->> { "user_id" : 0 } on :
                shard0000 Timestamp(2, 1)
            { "user_id" : 0 } -->> { "user_id" : 5999 } on : shard0000
                Timestamp(1, 3)
            { "user_id" : 5999 } -->> { "user_id" : 15999 } on : shard0001
                Timestamp(3, 1)
            { "user_id" : 15999 } -->> { "user_id" : 25999 } on : shard0002
                Timestamp(4, 1)
            { "user_id" : 25999 } -->> { "user_id" : 35999 } on : shard0003
                Timestamp(5, 1)
            { "user_id" : 35999 } -->> { "user_id" : 45999 } on : shard0004
                Timestamp(6, 1)
            { "user_id" : 45999 } -->> { "user_id" : 55999 } on : shard0005
                Timestamp(7, 1)
            { "user_id" : 55999 } -->> { "user_id" : 65999 } on : shard0006
                Timestamp(8, 1)
            { "user_id" : 65999 } -->> { "user_id" : 75999 } on : shard0007
                Timestamp(9, 1)
            { "user_id" : 75999 } -->> { "user_id" : 85999 } on : shard0008
                Timestamp(10, 1)
            { "user_id" : 85999 } -->> { "user_id" : 95999 } on : shard0009
                Timestamp(11, 1)
            { "user_id" : 95999 } -->> { "user_id" : { "$maxKey" : 1 } } on :
                shard0001 Timestamp(11, 0)
```

2. To which shard belongs the document having user_id: 55555 in the case of: 2, 5, and 10 shards? (2 marks)

Let's start with 10 shards. We can use `db.user.getShardDistribution()` to get exactly how many documents stored in each shard, so we can calculate that a doc with id 55555 will be stored on the 6th shard. However there is a command which shows us the exact value: `db.user.find({ user_id: 55555 }).explain()`. In the other two cases I will use `sh.status()` also to list the ranges.

In case of 10:

```
mongos> db.user.getShardDistribution()

Shard shard0000 at 127.0.0.1:27020
 data : 656KiB docs : 5999 chunks : 2
 estimated data per chunk : 328KiB
 estimated docs per chunk : 2999

Shard shard0001 at 127.0.0.1:27021
 data : 1.49MiB docs : 14001 chunks : 2
 estimated data per chunk : 765KiB
 estimated docs per chunk : 7000

Shard shard0002 at 127.0.0.1:27022
 data : 1.06MiB docs : 10000 chunks : 1
 estimated data per chunk : 1.06MiB
 estimated docs per chunk : 10000

Shard shard0003 at 127.0.0.1:27023
 data : 1.06MiB docs : 10000 chunks : 1
 estimated data per chunk : 1.06MiB
 estimated docs per chunk : 10000

Shard shard0004 at 127.0.0.1:27024
 data : 1.06MiB docs : 10000 chunks : 1
 estimated data per chunk : 1.06MiB
 estimated docs per chunk : 10000

Shard shard0005 at 127.0.0.1:27025
 data : 1.06MiB docs : 10000 chunks : 1
 estimated data per chunk : 1.06MiB
 estimated docs per chunk : 10000

Shard shard0006 at 127.0.0.1:27026
 data : 1.06MiB docs : 10000 chunks : 1
 estimated data per chunk : 1.06MiB
 estimated docs per chunk : 10000

Shard shard0007 at 127.0.0.1:27027
 data : 1.06MiB docs : 10000 chunks : 1
 estimated data per chunk : 1.06MiB
 estimated docs per chunk : 10000

Shard shard0008 at 127.0.0.1:27028
 data : 1.06MiB docs : 10000 chunks : 1
 estimated data per chunk : 1.06MiB
 estimated docs per chunk : 10000

Shard shard0009 at 127.0.0.1:27029
 data : 1.06MiB docs : 10000 chunks : 1
 estimated data per chunk : 1.06MiB
 estimated docs per chunk : 10000

Totals
 data : 10.68MiB docs : 100000 chunks : 12
 Shard shard0000 contains 5.99% data, 5.99% docs in cluster, avg obj size
    on shard : 112B
 Shard shard0001 contains 14% data, 14% docs in cluster, avg obj size on
    shard : 112B
 Shard shard0002 contains 10% data, 10% docs in cluster, avg obj size on
    shard : 112B
 Shard shard0003 contains 10% data, 10% docs in cluster, avg obj size on
    shard : 112B
 Shard shard0004 contains 10% data, 10% docs in cluster, avg obj size on
    shard : 112B
 Shard shard0005 contains 10% data, 10% docs in cluster, avg obj size on
    shard : 112B
 Shard shard0006 contains 10% data, 10% docs in cluster, avg obj size on
    shard : 112B
 Shard shard0007 contains 10% data, 10% docs in cluster, avg obj size on
    shard : 112B
 Shard shard0008 contains 10% data, 10% docs in cluster, avg obj size on
    shard : 112B
 Shard shard0009 contains 10% data, 10% docs in cluster, avg obj size on
    shard : 112B
```

```
mongos> db.user.find({ user_id: 55555 }).explain()
{
    "clusteredType" : "ParallelSort",
    "shards" : {
        "127.0.0.1:27025" : [
            {
                "cursor" : "BtreeCursor user_id_1",
                "isMultiKey" : false,
                "n" : 1,
                "nscannedObjects" : 1,
                "nscanned" : 1,
                "nscannedObjectsAllPlans" : 1,
                "nscannedAllPlans" : 1,
                "scanAndOrder" : false,
                "indexOnly" : false,
                "nYields" : 0,
                "nChunkSkips" : 0,
                "millis" : 0,
                "indexBounds" : {
                    "user_id" : [
                        [
                            55555,
                            55555
                        ]
                    ]
                },
                "server" : "regent.ecs.vuw.ac.nz:27025",
                "filterSet" : false
            }
        ]
    },
    "cursor" : "BtreeCursor user_id_1",
    "n" : 1,
    "nChunkSkips" : 0,
    "nYields" : 0,
    "nscanned" : 1,
    "nscannedAllPlans" : 1,
    "nscannedObjects" : 1,
    "nscannedObjectsAllPlans" : 1,
    "millisShardTotal" : 0,
    "millisShardAvg" : 0,
    "numQueries" : 1,
    "numShards" : 1,
    "indexBounds" : {
        "user_id" : [
            [
                55555,
                55555
            ]
        ]
    },
    "millis" : 0
}
```

Our document stored in the 6th shard:

```
Shard shard0005 at 127.0.0.1:27025
```

In case of 5:

In this case I use the `db.status()` to show where is the requested document.

```
sha-mongo status
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("59257444b71f43ba8836f521")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
    {  "_id" : "shard0002",  "host" : "127.0.0.1:27022" }
    {  "_id" : "shard0003",  "host" : "127.0.0.1:27023" }
    {  "_id" : "shard0004",  "host" : "127.0.0.1:27024" }
  databases:
    {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
    {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard0000" }
        mydb.user
            shard key: { "user_id" : 1 }
            chunks:
                shard0000   3
                shard0001   3
                shard0002   2
                shard0003   2
                shard0004   2
            { "user_id" : { "$minKey" : 1 } } -->> { "user_id" : 0 } on :
                shard0000 Timestamp(11, 1)
            { "user_id" : 0 } -->> { "user_id" : 5999 } on : shard0000
                Timestamp(1, 3)
            { "user_id" : 5999 } -->> { "user_id" : 15999 } on : shard0001
                Timestamp(7, 1)
            { "user_id" : 15999 } -->> { "user_id" : 25999 } on : shard0002
                Timestamp(8, 1)
            { "user_id" : 25999 } -->> { "user_id" : 35999 } on : shard0003
                Timestamp(9, 1)
            { "user_id" : 35999 } -->> { "user_id" : 45999 } on : shard0004
                Timestamp(10, 1)
            { "user_id" : 45999 } -->> { "user_id" : 55999 } on : shard0001
                Timestamp(6, 2)
            { "user_id" : 55999 } -->> { "user_id" : 65999 } on : shard0002
                Timestamp(7, 2)
            { "user_id" : 65999 } -->> { "user_id" : 75999 } on : shard0003
                Timestamp(8, 2)
            { "user_id" : 75999 } -->> { "user_id" : 85999 } on : shard0004
                Timestamp(9, 2)
            { "user_id" : 85999 } -->> { "user_id" : 95999 } on : shard0000
                Timestamp(10, 2)
            { "user_id" : 95999 } -->> { "user_id" : { "$maxKey" : 1 } } on :
                shard0001 Timestamp(11, 0)
```

```
mongos> db.user.find({ user_id: 55555 }).explain()
{
    "clusteredType" : "ParallelSort",
    "shards" : {
        "127.0.0.1:27021" : [
            {
                "cursor" : "BtreeCursor user_id_1",
                "isMultiKey" : false,
                "n" : 1,
                "nscannedObjects" : 1,
                "nscanned" : 1,
                "nscannedObjectsAllPlans" : 1,
                "nscannedAllPlans" : 1,
                "scanAndOrder" : false,
                "indexOnly" : false,
                "nYields" : 0,
                "nChunkSkips" : 0,
                "millis" : 0,
                "indexBounds" : {
                    "user_id" : [
                        [
                            55555,
                            55555
                        ]
                    ]
                },
                "server" : "regent.ecs.vuw.ac.nz:27021",
                "filterSet" : false
            }
        ]
    },
    "cursor" : "BtreeCursor user_id_1",
    "n" : 1,
    "nChunkSkips" : 0,
    "nYields" : 0,
    "nscanned" : 1,
    "nscannedAllPlans" : 1,
    "nscannedObjects" : 1,
    "nscannedObjectsAllPlans" : 1,
    "millisShardTotal" : 0,
    "millisShardAvg" : 0,
    "numQueries" : 1,
    "numShards" : 1,
    "indexBounds" : {
        "user_id" : [
            [
                55555,
                55555
            ]
        ]
    },
    "millis" : 2
}
```

Our document is on the 2nd shard, on `shard0001`.

```
{ "user_id" : 45999 } -->> { "user_id" : 55999 } on : shard0001 Timestamp(6, 2)
```

or `"shards" : { "127.0.0.1:27021" : [`

Lastly, let see where is our document in case of 2 shards. Our document is on 2nd shard. See the result below.

```
mongos> db.user.find({ user_id: 55555 }).explain()
```

```
{
    "clusteredType" : "ParallelSort",
    "shards" : {
        "127.0.0.1:27021" : [
            {
                "cursor" : "BtreeCursor user_id_1",
                "isMultiKey" : false,
                "n" : 1,
                "nscannedObjects" : 1,
                "nscanned" : 1,
                "nscannedObjectsAllPlans" : 1,
                "nscannedAllPlans" : 1,
                "scanAndOrder" : false,
                "indexOnly" : false,
                "nYields" : 0,
                "nChunkSkips" : 0,
                "millis" : 0,
                "indexBounds" : {
                    "user_id" : [
                        [
                            55555,
                            55555
                        ]
                    ]
                },
                "server" : "regent.ecs.vuw.ac.nz:27021",
                "filterSet" : false
            }
        ]
    },
    "cursor" : "BtreeCursor user_id_1",
    "n" : 1,
    "nChunkSkips" : 0,
    "nYields" : 0,
    "nscanned" : 1,
    "nscannedAllPlans" : 1,
    "nscannedObjects" : 1,
    "nscannedObjectsAllPlans" : 1,
    "millisShardTotal" : 0,
    "millisShardAvg" : 0,
    "numQueries" : 1,
    "numShards" : 1,
    "indexBounds" : {
        "user_id" : [
            [
                55555,
                55555
            ]
        ]
    },
    "millis" : 1
}

mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("592576537eba76d77cd1d91a")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
  databases:
    {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
    {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard0000" }
        mydb.user
            shard key: { "user_id" : 1 }
            chunks:
                shard0001   6
                shard0000   6
            { "user_id" : { "$minKey" : 1 } } -->> { "user_id" : 0 } on :
                shard0001 Timestamp(12, 1)
            { "user_id" : 0 } -->> { "user_id" : 5999 } on : shard0000
                Timestamp(11, 1)
            { "user_id" : 5999 } -->> { "user_id" : 15999 } on : shard0001
                Timestamp(5, 1)
            { "user_id" : 15999 } -->> { "user_id" : 25999 } on : shard0000
                Timestamp(3, 2)
            { "user_id" : 25999 } -->> { "user_id" : 35999 } on : shard0001
                Timestamp(4, 2)
            { "user_id" : 35999 } -->> { "user_id" : 47999 } on : shard0000
                Timestamp(6, 2)
            { "user_id" : 47999 } -->> { "user_id" : 57999 } on : shard0001
                Timestamp(7, 2)
            { "user_id" : 57999 } -->> { "user_id" : 67999 } on : shard0000
                Timestamp(8, 2)
            { "user_id" : 67999 } -->> { "user_id" : 77999 } on : shard0001
                Timestamp(9, 2)
            { "user_id" : 77999 } -->> { "user_id" : 87999 } on : shard0000
                Timestamp(10, 2)
            { "user_id" : 87999 } -->> { "user_id" : 97999 } on : shard0001
                Timestamp(11, 2)
            { "user_id" : 97999 } -->> { "user_id" : { "$maxKey" : 1 } } on :
                shard0000 Timestamp(12, 0)
```

c) Use the configuration with 10 shards. Connect to the mongo shell of the server storing the shard that contains the user document with user_id: 55555. (2 marks)

Our server: `127.0.0.1:27025`

```
mongos> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("59257942d7bcff829837a8c6")
}
  shards:
    {  "_id" : "shard0000",  "host" : "127.0.0.1:27020" }
    {  "_id" : "shard0001",  "host" : "127.0.0.1:27021" }
    {  "_id" : "shard0002",  "host" : "127.0.0.1:27022" }
    {  "_id" : "shard0003",  "host" : "127.0.0.1:27023" }
    {  "_id" : "shard0004",  "host" : "127.0.0.1:27024" }
    {  "_id" : "shard0005",  "host" : "127.0.0.1:27025" }
    {  "_id" : "shard0006",  "host" : "127.0.0.1:27026" }
    {  "_id" : "shard0007",  "host" : "127.0.0.1:27027" }
    {  "_id" : "shard0008",  "host" : "127.0.0.1:27028" }
    {  "_id" : "shard0009",  "host" : "127.0.0.1:27029" }
  databases:
    {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
    {  "_id" : "mydb",  "partitioned" : true,  "primary" : "shard0000" }
        mydb.user
            shard key: { "user_id" : 1 }
            chunks:
                shard0000   2
                shard0001   2
                shard0002   1
                shard0003   1
                shard0004   1
                shard0005   1
                shard0006   1
                shard0007   1
                shard0008   1
                shard0009   1
            { "user_id" : { "$minKey" : 1 } } -->> { "user_id" : 0 } on :
                shard0000 Timestamp(2, 1)
            { "user_id" : 0 } -->> { "user_id" : 5999 } on : shard0000
                Timestamp(1, 3)
            { "user_id" : 5999 } -->> { "user_id" : 15999 } on : shard0001
                Timestamp(3, 1)
            { "user_id" : 15999 } -->> { "user_id" : 25999 } on : shard0002
                Timestamp(4, 1)
            { "user_id" : 25999 } -->> { "user_id" : 35999 } on : shard0003
                Timestamp(5, 1)
            { "user_id" : 35999 } -->> { "user_id" : 45999 } on : shard0004
                Timestamp(6, 1)
            { "user_id" : 45999 } -->> { "user_id" : 55999 } on : shard0005
                Timestamp(7, 1)
            { "user_id" : 55999 } -->> { "user_id" : 65999 } on : shard0006
                Timestamp(8, 1)
            { "user_id" : 65999 } -->> { "user_id" : 75999 } on : shard0007
                Timestamp(9, 1)
            { "user_id" : 75999 } -->> { "user_id" : 85999 } on : shard0008
                Timestamp(10, 1)
            { "user_id" : 85999 } -->> { "user_id" : 95999 } on : shard0009
                Timestamp(11, 1)
            { "user_id" : 95999 } -->> { "user_id" : { "$maxKey" : 1 } } on :
                shard0001 Timestamp(11, 0)
```

```
$ sha-mongo connect 5
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27025/test
> use mydb
```

i. Retrieve the document with user_id: 55555.

```
> db.user.find({user_id: 55555})
{ "_id" : ObjectId("592579512a56d195595d2931"), "user_id" : 55555, "name" :
    "Kristina", "number" : 1198 }
```

ii. Retrieve the document with user_id: 1.

```
> db.user.find({user_id: 1})
```

No result in this case.

d) Use the configuration with 10 shards. Connect now to the shell of the mongos process. (2 marks)

```
$ sha-mongo connect
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
> use mydb
```

i. Retrieve the document with `user_id: 55555`.

```
mongos> db.user.find({user_id: 55555})
{ "_id" : ObjectId("592579512a56d195595d2931"), "user_id" : 55555, "name" :
    "Kristina", "number" : 1198 }
```

ii. Retrieve the document with `user_id: 1`.

```
mongos> db.user.find({user_id: 1})
{ "_id" : ObjectId("5925794e2a56d195595c502f"), "user_id" : 1, "name" :
    "Jeff", "number" : 7993 }
```

e) Explain MongoDB behavior in questions c) and d) above. (4 marks)

MongoDB uses our main Config Server for storing the cluster's metadata and the map for the cluster's data set to shards. The Application Server uses Routers (mongos) to find documents in the cluster.

Because in question c) we connected to a shard server directly, it cannot see outside of own dataset. That shard had the record with id `55555`, we were able to retrieve, however the document with id 1 was not on that server. The document with id `1` doesn't exists for that shard.

On the other side, when we connected with `mongos` command to the `Router` and the `Application Server`, we had access to the `Config Servers` also, so we were able to find any document across the cluster on each shard, so we were able to retrieve documents with id `1` and with id `55555`.

f) Stop mongod server storing the shard that contains the document with user_id: `55555` and connect to the shell of the mongos process, again. (8 marks)

```
$ sha-mongo stop 5
Stopping mongod --port 27025 shadb5
$ sha-mongo connect
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
mongos> use mydb
```

i. Retrieve the document with `user_id: 55555`.

```
mongos> db.user.find({user_id: 55555})
error: {
    "$err" : "socket exception [CONNECT_ERROR] for 127.0.0.1:27025",
    "code" : 11002,
    "shard" : "shard0005"
}
```

ii. Retrieve the document with `user_id: 1`.

```
mongos> db.user.find({user_id: 1})
{ "_id" : ObjectId("5925794e2a56d195595c502f"), "user_id" : 1, "name" :
    "Jeff", "number" : 7993 }
```

iii. What percentage (roughly) of your database became unavailable? Will it become available again if you restart the mongod server?

We can see with `sh.status()` which range is stored on the 6th server:
`{ "user_id" : 45999 } -->> { "user_id" : 55999 } on : shard0005 Timestamp(7, 1)`

Calculation:

```
55999-45999 = 10000
10000 / 100000 = 0.1
10% of the database is unavailable.
```

Let’s have more experiment with our data:

```
mongos> db.user.find({user_id: 45998})
{ "_id" : ObjectId("592579502a56d195595d03dc"), "user_id" : 45998, "name" :
    "Bill", "number" : 7914 }
mongos> db.user.find({user_id: 45999})
error: {
    "$err" : "socket exception [CONNECT_ERROR] for 127.0.0.1:27025",
    "code" : 11002,
    "shard" : "shard0005"
}
mongos> db.user.find({user_id: 55998})
error: {
    "$err" : "socket exception [CONNECT_ERROR] for 127.0.0.1:27025",
    "code" : 11002,
    "shard" : "shard0005"
}
mongos> db.user.find({user_id: 55999})
{ "_id" : ObjectId("592579512a56d195595d2aed"), "user_id" : 55999, "name" :
    "Katie", "number" : 5064 }
```

```
$ sha-mongo start 5
Starting mongod --port 27025 shadb5
about to fork child process, waiting until server is ready for connections.
forked process: 21997
child process started successfully, parent exiting
$ sha-mongo connect
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
mongos> use mydb
switched to db mydb
mongos> db.user.find({user_id: 55555})
{ "_id" : ObjectId("592579512a56d195595d2931"), "user_id" : 55555, "name" :
    "Kristina", "number" : 1198 }
mongos> db.user.find({user_id: 55998})
{ "_id" : ObjectId("592579512a56d195595d2aec"), "user_id" : 55998, "name" :
    "Eliot", "number" : 2627 }
```

Yes, we can retrieve our data again after we restarted the server.

Question 7. Sharding and Replication

(16 marks)

Use the `sharep-mongo` script to produce a deployment with `2` shards having `3` replicas each. Populate your replica sets by the `user` collection using the `test` parameter to the sharep-mongo script.

In this question, you will need to get information about the status of your replication sets (e.g. ports of master and slave servers, which servers are up and which down). Unhappily, the `sharep-mongo` script does not offer you this option. So you will need to do it manually. To get information about the status of a replica set, connect to the mongo shell of any of the replica servers, switch to the database of interest, and type `rs.status()`.

```
$ sharep-mongo init
$ sharep-mongo test
$ sharep-mongo status
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "version" : 4,
    "minCompatibleVersion" : 4,
    "currentVersion" : 5,
    "clusterId" : ObjectId("5926a71307269e9ed9d0ec0a")
  }
  shards:
    {  "_id" : "rs0",  "host" : "rs0/127.0.0.1:27020,127.0.0.1:27021,
        127.0.0.1:27022" }
    {  "_id" : "rs1",  "host" : "rs1/127.0.0.1:27023,127.0.0.1:27024,
        127.0.0.1:27025" }
  databases:
    {  "_id" : "admin",  "partitioned" : false,  "primary" : "config" }
    {  "_id" : "mydb",  "partitioned" : true,  "primary" : "rs0" }
        mydb.user
            shard key: { "user_id" : 1 }
            chunks:
                rs0 5
                rs1 6
            { "user_id" : { "$minKey" : 1 } } -->> { "user_id" : 0 } on :
                rs0 Timestamp(4, 1)
            { "user_id" : 0 } -->> { "user_id" : 4999 } on : rs0 Timestamp(1, 3)
            { "user_id" : 4999 } -->> { "user_id" : 15999 } on : rs1
                Timestamp(3, 1)
            { "user_id" : 15999 } -->> { "user_id" : 27999 } on : rs1
                Timestamp(2, 4)
            { "user_id" : 27999 } -->> { "user_id" : 39999 } on : rs1
                Timestamp(2, 6)
            { "user_id" : 39999 } -->> { "user_id" : 51999 } on : rs1
                Timestamp(2, 8)
            { "user_id" : 51999 } -->> { "user_id" : 63999 } on : rs1
                Timestamp(2, 10)
            { "user_id" : 63999 } -->> { "user_id" : 74999 } on : rs0
                Timestamp(3, 2)
            { "user_id" : 74999 } -->> { "user_id" : 86999 } on : rs0
                Timestamp(3, 4)
            { "user_id" : 86999 } -->> { "user_id" : 98999 } on : rs0
                Timestamp(3, 6)
            { "user_id" : 98999 } -->> { "user_id" : { "$maxKey" : 1 } } on :
                rs1 Timestamp(4, 0)
```

a) (2 mark) Find the port number of the master server of the replica set rs0. The port number is the last five digits of the value of the server's name field.

See the `rs.status()` result below. Name field: `"name" : "127.0.0.1:27020"`, so the requested port number: `27020`

```
$ sharep-mongo connect 0 0
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27020/test
rs0:PRIMARY> rs.status()
  {
    "set" : "rs0",
    "date" : ISODate("2017-05-25T09:13:07Z"),
    "myState" : 1,
    "members" : [
        {
            "_id" : 0,
            "name" : "127.0.0.1:27020",
            "health" : 1,
            "state" : 1,
            "stateStr" : "PRIMARY",
            "uptime" : 222,
            "optime" : Timestamp(1495703523, 723),
            "optimeDate" : ISODate("2017-05-25T09:12:03Z"),
            "electionTime" : Timestamp(1495703374, 1),
            "electionDate" : ISODate("2017-05-25T09:09:34Z"),
            "self" : true
        },
        {
            "_id" : 1,
            "name" : "127.0.0.1:27021",
            "health" : 1,
            "state" : 2,
            "stateStr" : "SECONDARY",
            "uptime" : 221,
            "optime" : Timestamp(1495703523, 723),
            "optimeDate" : ISODate("2017-05-25T09:12:03Z"),
            "lastHeartbeat" : ISODate("2017-05-25T09:13:06Z"),
            "lastHeartbeatRecv" : ISODate("2017-05-25T09:13:07Z"),
            "pingMs" : 0,
            "syncingTo" : "127.0.0.1:27020"
        },
        {
            "_id" : 2,
            "name" : "127.0.0.1:27022",
            "health" : 1,
            "state" : 2,
            "stateStr" : "SECONDARY",
            "uptime" : 221,
            "optime" : Timestamp(1495703523, 723),
            "optimeDate" : ISODate("2017-05-25T09:12:03Z"),
            "lastHeartbeat" : ISODate("2017-05-25T09:13:06Z"),
            "lastHeartbeatRecv" : ISODate("2017-05-25T09:13:07Z"),
            "pingMs" : 0,
            "syncingTo" : "127.0.0.1:27020"
        }
    ],
    "ok" : 1
}
```

b) (2 mark) Connect to the master server of the replica set `rs0` .

```
$ sharep-mongo connect 0 0
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27020/test
rs0:PRIMARY>
```

i. Retrieve the document having `user_id: 1` from the `mydb.user` collection. (1 mark)

```
rs0:PRIMARY> use mydb
switched to db mydb
rs0:PRIMARY> db.user.find({user_id: 1})
{ "_id" : ObjectId("5926a74884b79f6769eef781"), "user_id" : 1,
    "name" : "Matt", "number" : 1556 }
```

ii. Insert the document `{"user_id": 100000, "name": "Steve", "number": 0}` into the `mydb.user` collection. (1 mark)

```
rs0:PRIMARY> db.user.insert({"user_id": 100000, "name": "Steve", "number": 0})
WriteResult({ "nInserted" : 1 })
```

c) (2 mark) Connect to a slave server of the replica set `rs0` .

```
$ sharep-mongo connect 0 1
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27021/test
```

i. Retrieve the document having `user_id: 1` from the `mydb.user` collection. (1 mark)

```
rs0:SECONDARY> use mydb
switched to db mydb
rs0:SECONDARY> db.user.find({user_id: 1})
error: { "$err" : "not master and slaveOk=false", "code" : 13435 }
```

ii. Insert the document `{"user_id": 100001, "name": "Steve", "number": 1}` into the `mydb.user` collection. (1 mark)

```
rs0:SECONDARY> db.user.insert({"user_id": 100001, "name": "Steve", "number": 1})
WriteResult({ "writeError" : { "code" : undefined, "errmsg" : "not master" } })
```

d) (2 mark) Stop the master server of the replica set rs0.

```
$ sharep-mongo stop 0 0
Stopping mongod --port 27020 sharep-rs0-0
```

i. View the status of the replica set rs0 and describe it briefly. (1 mark)

```
$ sharep-mongo connect 0 0
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27020/test
2017-05-25T21:54:58.345+1200 warning: Failed to connect to 127.0.0.1:27020,
    reason: errno:111 Connection refused
2017-05-25T21:54:58.345+1200 Error: couldn't connect to server 127.0.0.1:27020
    (127.0.0.1), connection attempt failed at src/mongo/shell/mongo.js:148
exception: connect failed

$ sharep-mongo connect 0 1
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27021/test
rs0:SECONDARY> rs.status()
{
    "set" : "rs0",
    "date" : ISODate("2017-05-25T09:56:38Z"),
    "myState" : 2,
    "syncingTo" : "127.0.0.1:27022",
    "members" : [
        {
            "_id" : 0,
            "name" : "127.0.0.1:27020",
            "health" : 0,
            "state" : 8,
            "stateStr" : "(not reachable/healthy)",
            "uptime" : 0,
            "optime" : Timestamp(1495705552, 1),
            "optimeDate" : ISODate("2017-05-25T09:45:52Z"),
            "lastHeartbeat" : ISODate("2017-05-25T09:56:37Z"),
            "lastHeartbeatRecv" : ISODate("2017-05-25T09:53:26Z"),
            "pingMs" : 0
        },
        {
            "_id" : 1,
            "name" : "127.0.0.1:27021",
            "health" : 1,
            "state" : 2,
            "stateStr" : "SECONDARY",
            "uptime" : 835,
            "optime" : Timestamp(1495705552, 1),
            "optimeDate" : ISODate("2017-05-25T09:45:52Z"),
            "self" : true
        },
        {
            "_id" : 2,
            "name" : "127.0.0.1:27022",
            "health" : 1,
            "state" : 1,
            "stateStr" : "PRIMARY",
            "uptime" : 832,
            "optime" : Timestamp(1495705552, 1),
            "optimeDate" : ISODate("2017-05-25T09:45:52Z"),
            "lastHeartbeat" : ISODate("2017-05-25T09:56:36Z"),
            "lastHeartbeatRecv" : ISODate("2017-05-25T09:56:36Z"),
            "pingMs" : 0,
            "electionTime" : Timestamp(1495706011, 1),
            "electionDate" : ISODate("2017-05-25T09:53:31Z")
        }
    ],
    "ok" : 1
}
```

Our previous master (PRIMARY) is down. We can see the status of this replica set if we connect to an other server on this shard. Clearly, our server on port `27022` became the master (PRIMARY). Server on port `27020` is `not reachable/healthy`. Our server on port `27021` is still our slave (SECONDARY).

We can see details about heartbeats and information about `electionTime` and `electionDate`. Our server `27022` was elected because it was probably faster than the other secondary server.

Interesting to see, that the last heartbeat from our dead server received at `"lastHeartbeatRecv" : ISODate("2017-05-25T09:53:26Z"),` and our new master was elected at `"electionDate" : ISODate("2017-05-25T09:53:31Z")`. `09:53:26` vs `09:53:31`. There was only `5` seconds long interval without master.

Please note, we still have this option here: `"syncingTo" : "127.0.0.1:27022",` , this option won't be there when we will stop an other server.

ii. Connect to the `mongos` shell. Retrieve the document with `user_id: 1` from the `mydb.user` collection. Insert the document `{"user_id": 100001, "name": "Steve", "number": 1}` into the mydb.user collection. (1 mark)

```
$ sharep-mongo connect
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
mongos> show dbs
admin   (empty)
config  0.016GB
mydb    0.063GB
mongos> use mydb
switched to db mydb
mongos> show collections
system.indexes
user
mongos> db.user.find({user_id:1})
{ "_id" : ObjectId("5926a74884b79f6769eef781"), "user_id" : 1, "name" : "Matt",
    "number" : 1556 }
mongos> db.user.insert({"user_id": 100001, "name": "Steve", "number": 1})
WriteResult({ "nInserted" : 1 })
```

Little experiment with `.explain()` :

```
mongos> db.user.find({user_id: 1}).explain()
{
    "clusteredType" : "ParallelSort",
    "shards" : {
        "rs0/127.0.0.1:27020,127.0.0.1:27021,127.0.0.1:27022" : [
            {
                "cursor" : "BtreeCursor user_id_1",
                "isMultiKey" : false,
                "n" : 1,
                "nscannedObjects" : 1,
                "nscanned" : 1,
                "nscannedObjectsAllPlans" : 1,
                "nscannedAllPlans" : 1,
                "scanAndOrder" : false,
                "indexOnly" : false,
                "nYields" : 0,
                "nChunkSkips" : 0,
                "millis" : 0,
                "indexBounds" : {
                    "user_id" : [
                        [
                            1,
                            1
                        ]
                    ]
                },
                "server" : "regent.ecs.vuw.ac.nz:27022",
                "filterSet" : false
            }
        ]
    },
    "cursor" : "BtreeCursor user_id_1",
    "n" : 1,
    "nChunkSkips" : 0,
    "nYields" : 0,
    "nscanned" : 1,
    "nscannedAllPlans" : 1,
    "nscannedObjects" : 1,
    "nscannedObjectsAllPlans" : 1,
    "millisShardTotal" : 0,
    "millisShardAvg" : 0,
    "numQueries" : 1,
    "numShards" : 1,
    "indexBounds" : {
        "user_id" : [
            [
                1,
                1
            ]
        ]
    },
    "millis" : 0
}

mongos> db.user.find({user_id: 100001}).explain()
{
    "clusteredType" : "ParallelSort",
    "shards" : {
        "rs1/127.0.0.1:27023,127.0.0.1:27024,127.0.0.1:27025" : [
            {
                "cursor" : "BtreeCursor user_id_1",
                "isMultiKey" : false,
                "n" : 1,
                "nscannedObjects" : 1,
                "nscanned" : 1,
                "nscannedObjectsAllPlans" : 1,
                "nscannedAllPlans" : 1,
                "scanAndOrder" : false,
                "indexOnly" : false,
                "nYields" : 0,
                "nChunkSkips" : 0,
                "millis" : 0,
                "indexBounds" : {
                    "user_id" : [
                        [
                            100001,
                            100001
                        ]
                    ]
                },
                "server" : "regent.ecs.vuw.ac.nz:27023",
                "filterSet" : false
            }
        ]
    },
    "cursor" : "BtreeCursor user_id_1",
    "n" : 1,
    "nChunkSkips" : 0,
    "nYields" : 0,
    "nscanned" : 1,
    "nscannedAllPlans" : 1,
    "nscannedObjects" : 1,
    "nscannedObjectsAllPlans" : 1,
    "millisShardTotal" : 0,
    "millisShardAvg" : 0,
    "numQueries" : 1,
    "numShards" : 1,
    "indexBounds" : {
        "user_id" : [
            [
                100001,
                100001
            ]
        ]
    },
    "millis" : 0
}
```

The first document request (id:1) retrieved from the `rs0` new master server (`27022`). The second request (id:100001) retrieved from `rs1` master (`27023`).

e) (2 mark) Stop the remaining slave server of the replica set rs0.

```
$ sharep-mongo stop 0 1
Stopping mongod --port 27021 sharep-rs0-1
```

i. View the status of the replica set rs0 and describe it briefly. (1 mark)

```
sharep-mongo connect 0 0
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27020/test
2017-05-25T22:10:09.473+1200 warning: Failed to connect to 127.0.0.1:27020,
    reason: errno:111 Connection refused
2017-05-25T22:10:09.473+1200 Error: couldn't connect to server 127.0.0.1:27020
    (127.0.0.1), connection attempt failed at src/mongo/shell/mongo.js:148
exception: connect failed

$ sharep-mongo connect 0 1
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27021/test
2017-05-25T22:10:12.763+1200 warning: Failed to connect to 127.0.0.1:27021,
    reason: errno:111 Connection refused
2017-05-25T22:10:12.763+1200 Error: couldn't connect to server 127.0.0.1:27021
    (127.0.0.1), connection attempt failed at src/mongo/shell/mongo.js:148
exception: connect failed

$ sharep-mongo connect 0 2
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27022/test
rs0:SECONDARY> rs.status()
{
    "set" : "rs0",
    "date" : ISODate("2017-05-25T10:10:22Z"),
    "myState" : 2,
    "members" : [
        {
            "_id" : 0,
            "name" : "127.0.0.1:27020",
            "health" : 0,
            "state" : 8,
            "stateStr" : "(not reachable/healthy)",
            "uptime" : 0,
            "optime" : Timestamp(1495705552, 1),
            "optimeDate" : ISODate("2017-05-25T09:45:52Z"),
            "lastHeartbeat" : ISODate("2017-05-25T10:10:21Z"),
            "lastHeartbeatRecv" : ISODate("2017-05-25T09:53:26Z"),
            "pingMs" : 0
        },
        {
            "_id" : 1,
            "name" : "127.0.0.1:27021",
            "health" : 0,
            "state" : 8,
            "stateStr" : "(not reachable/healthy)",
            "uptime" : 0,
            "optime" : Timestamp(1495705552, 1),
            "optimeDate" : ISODate("2017-05-25T09:45:52Z"),
            "lastHeartbeat" : ISODate("2017-05-25T10:10:21Z"),
            "lastHeartbeatRecv" : ISODate("2017-05-25T10:09:12Z"),
            "pingMs" : 0,
            "syncingTo" : "127.0.0.1:27022"
        },
        {
            "_id" : 2,
            "name" : "127.0.0.1:27022",
            "health" : 1,
            "state" : 2,
            "stateStr" : "SECONDARY",
            "uptime" : 1659,
            "optime" : Timestamp(1495705552, 1),
            "optimeDate" : ISODate("2017-05-25T09:45:52Z"),
            "self" : true
        }
    ],
    "ok" : 1
}
```

We can see from the status, `27020` and `27021` are down, so we lost our previous slave server on this shard. `27022` was our master server a moment ago, but it became now a slave (SECONDARY) server again. We don't have master server anymore on this replica set. We can clearly see here what we learned: "In a three members replica set, if two members are unavailable, the remaining one remains, or becomes the secondary disregarding what role it had before."

We can see details about heartbeats also, what time was it sent and received from servers.

Please note, we don't have `"syncingTo"` field in our status.

ii. Connect to the `mongos` shell. Retrieve the document with `user_id: 1` from the `mydb.user` collection. (1 mark)

```
$ sharep-mongo connect
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27017/test
mongos> use mydb
switched to db mydb
mongos> db.user.find({user_id: 1})
error: {
    "$err" : "ReplicaSetMonitor no master found for set: rs0",
    "code" : 10009,
    "shard" : "rs0"
}
```

Now, we can see again in the error message, no master left in `rs0`. It also means, that if read from slave not allowed, than our replica set is try to be `Strictly Consistent`. (If clients are allowed to read from secondaries, the replica set provides an `Eventual Consistency`. We can turn on this feature using `rs.slaveOk()` method. Ref: https://docs.mongodb.com/v2.6/reference/method/rs.slaveOk/)

f) (6 marks) Briefly describe what you have learned by doing subquestions b), c), d), and e) of question 7.

Please read my comments in b), c), d) and e) above.

Summary:

- We started our experiment with 2 replica sets. We had one master (`27020`) and two slaves (`27021`, `27022`) in our first replica set (`rs0`).
- When we connected to our master server (in task `b`), it was able to read and write data. (Master server accepts all write request from clients, updates its data set and records update operations in its operation log.)
- When we connected to our slave server (in task `c`), it was not able to read or write any data, because a slave server can receive operations from the primary oplog only and apply them on their data sets.
- When we stopped our master server an "election" happened and we had a new master in our `rs0` replica set. (Please read my comments in question `d` and `c`.)
- Because we had a new master server, we were able to read and write data, when we connected to this new master.
- Our replica set try to be `Strictly Consistent` because after no master left, we were not able to read or write any data in that replica set. In this case our data set still can be `Eventually Consistent`, but we have to setup it with `rs.slaveOk()`. (More details in question `e`.)