School of Engineering and Computer Science

SWEN 432
Advanced Database Design and Implementation

Assignment 4

Due date: Monday 29 May at 11:59 pm

The objective of this assignment is to test your understanding of MongoDB pipeline aggregates method, sharding and replication.

The Assignment is worth 5.0% of your final grade. The Assignment is marked out of 100.

When doing the assignment you will need to use MongoDB, the cloud database management system. There is a brief Instruction containing all needed commands given in an Appendix to this Assignment. Read it carefully before starting doing Assignment. In the course of answering assignment questions, you will need to issue command prompt instructions and to use a MongoDB shell of:

- A single sharded,
- A multisharded, and
- A multisharded and replicated

MongoDB installation. For deployment of each of these installations, scripts are provided in the Instruction.

There are two parts to this assignment. The first one is about using (pipeline) `aggregate()` methods in the mongo shell of a single sharded deployment. The collection documents for the first part contain data about marinas, boats, sailors, and boat reservations made by sailors. These documents are contained in the file

`ass4_reserves_17.json`

Import the file above into a collection with the name `reserves`. Note, the file `ass4_reserves_17.json` contains data that slightly differ from your `reserves` collection in Assignment 3.

Important

In your answers to the assignment questions, show the commands and methods you issued and used and the answers produced by MongoDB on the standard output.

Part I [62 marks]

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

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([{$match: {'reserves.sailor':
{$exists: true, $ne: null}}},
{$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

```
db.reserves.aggregate([{$match: {'reserves.date': {$exists:
  true, $ne: null}}},
{$group: {_id: {sailorId: "$reserves.sailor.sailorId",
  name: "$reserves.sailor.name", address:
  "$reserves.sailor.address"},
no_of_reserves: {$sum: 1} }}, {$sort: {no_of_reserves: -
  1}}, {$limit: 1}])

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

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

```
db.reserves.aggregate([{$match:        {'reserves.res_date': {$exists: true}}}, {$group: {_id: 0, total_reserves: {$sum: 1} }}, {$project: {_id: 0, total_reserves: 1}}])
{ "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

Note:
There are 20 documents in the `reserves` collection, of these:

- 18 documents contain valid reservations,
- 19 documents contain sailors,
- 1 document contains a sailor with no reservation, and
- 1 document contains a boat with no reservation.

There are exactly 7 sailors.
Accordingly, the average requested is 18/7 = 2.5714…

The following query returns a wrong average since it does not take into account the sailor who did not make any reservations:

```
> db.reserves.aggregate([{$match: {'reserves.res_date':
  {$exists: true}}},
{$group: {_id: "$reserves.sailor.sailorId", no_of_reserves:
{$sum: 1} }},
{$group: {_id: 0, avg_reserves: {$avg:
"$no_of_reserves"}}}])
{ "_id" : 0, "avg_reserves" : 3 }
```

The following query returns a wrong average since the expression `{$sum: 1}` counts documents containing sailors and that leads to a wrong total number of 19 reserves:

```
> db.reserves.aggregate([{$match: {'reserves.sailor':
  {$exists: true}}},
.{$group: {_id: "$reserves.sailor.sailorId",
  no_of_reserves: {$sum: 1} }},
 {$group: {_id: 0, avg_reserves: {$avg:
  "$no_of_reserves"}}}])
{ "_id" : 0, "avg_reserves" : 2.7142857142857144 }
```

The two step approach performs computing of the total number of reserves as in question 3. above, and uses this number to compute the average:

```
var total_reserves = db.reserves.aggregate([{$match:
{'reserves.res_date': {$exists: true}}},

{$group: {_id: 0, total_reserves: {$sum: 1} }},

{$project: {_id: 0, total_reserves: 1}}])

> no_of_reserves = total_reserves.next()

{ "total_reserves" : 18 }

> db.reserves.aggregate([{$match: {'reserves.sailor':
{$exists: true, $ne: null}}},

{$group: {_id: {sailorId: "$reserves.sailor.sailorId"}}},

{$group: {_id: 0, no_of_sailors: {$sum: 1}} },

{$project: {_id: 0, avg_reserves: {$divide:
  [no_of_reserves.total_reserves, "$no_of_sailors"]}}}])

{ "avg_reserves" : 2.5714285714285716 }
```

Here is a single multistage statement that computes: a correct average:

```
> db.reserves.aggregate([

{ $match: {"reserves.sailor.sailorId": {$exists: true, $ne:
null} } },

{ $group: { _id: "$reserves.sailor.sailorId",
"reserves_per_person": { "$sum": { "$cond":
["$reserves.res_date", 1, 0] } } } },

{ $group: { _id: null, "total_avg": { $avg:
"$reserves_per_person" } } },

{ $project: { total_avg: 1, _id: 0 } }

])

{ "total_avg" : 2.5714285714285716 }
```

Question 5. **[10 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.

ANSWER

```
{
var sailor_name = "Paul"
var sailor_skills =
db.reserves.distinct('reserves.sailor.skills',
{'reserves.sailor.name': sailor_name})
print(sailor_name + " is qualified to drive the following
boats:");

db.reserves.aggregate([{$match: {'reserves.boat.driven_by':
{$exists: true, $ne: null} } },
{$group: { _id: {marina: '$marina.name', boat:
'$reserves.boat.number'}, driven_by: {$first:
'$reserves.boat.driven_by'}, name: {$first:
'$reserves.boat.name'}} },
{$project: { _id: 0, boat: { $cond: { if: { $setIsSubset:
['$driven_by', sailor_skills] }, then:
{marina: '$_id.marina', number: '$_id.boat', name:'$name'},
else: null}}}},
{$match: { boat: { $ne: null } } } ] )
}

Paul is qualified to drive the following boats:
{ "boat" : { "marina" : "Evans Bay", "number" : 137, "name"
: "Sea Gull" } }
{ "boat" : { "marina" : "Port Nickolson", "number" : 137,
"name" : "Sea Gull" } }
{ "boat" : { "marina" : "Port Nicholson", "number" : 818,
"name" : "Night Breeze" } }
{ "boat" : { "marina" : "Port Nicholson", "number" : 131,
"name" : "Penguin" } }
{ "boat" : { "marina" : "Sea View", "number" : 111, "name"
: "Killer Whale" } }
```

Bonus 15 marks question

Use `aggregate()` method and java scripts 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

```
var myavg = db.reserves.aggregate([
{ $match: {"reserves.sailor.sailorId": {$exists: true, $ne:
null}} },
{ $project: {"sailorId": "$reserves.sailor.sailorId",
"res_date": {$ifNull: ["$reserves.res_date", "Unspecified"]
}} },
{ $group: { _id: "$sailorId", "totalPerPerson": { "$sum": {
"$cond": { if: {"$eq" : [ "$res_date", "Unspecified" ]},
then : 0, else : 1 }} } } },
{ $group: { _id: null, "total_avg": { $avg:
"$totalPerPerson" } } },
{ $project: { total_avg: 1, _id: 0 } }
])

var my_avg = myavg.next()

db.reserves.aggregate([
{$match: {'reserves.sailor.sailorId': {$exists: true, $ne:
null}, 'reserves.res_date': {$exists: true, $ne: null}}},
{$group: {_id: '$reserves.sailor', sum_per_sailor: {$sum:
1}}},
{$project: {_id: 0, sailor_name: {$cond: {if: {$gt:
['$sum_per_sailor', my_avg.total_avg]}, then : '$_id.name',
else: null}}, reserves: '$sum_per_sailor'}}, {$match:
{sailor_name: {$ne: null}}}])
{ "sailor_name" : "Milan", "reserves" : 6 }
{ "sailor_name" : "James", "reserves" : 3 }
{ "sailor_name" : "Peter", "reserves" : 3 }
```

Part II [38 marks]

Before you start answering part two questions, read carefully Instruction in the Appendix. In this part, you are going to make a number of MongoDB deployments having different numbers of shards. One of deployments is going to be both sharded and replicated. You will experiment with these deployments and the knowledge gained by experimenting will help you answer the questions.

Note: Each MongoDB occupies a few hundreds of GB on disk. To avoid using disk space in vain, whenever you finish experimenting with a deployment,

perform a `cleanall` command (have a look in the Instruction for the `cleanall` command).

Question 6. Sharding **[22 marks]**

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.

a) **[2 marks]** Which partitioning method has been used for sharding the `user` collection?

ANSWER

Range based sharding

b) **[4 marks]**

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

ANSWER

The following commands return the number of chunks per shard:

```
> use mydb
> db.user.getShardDistribution()
```

2 shards: 6, 5 shards: 2, 10 shards: 1.2

ii. To which shard belongs the document having `user_id: 55555` in the case of: 2, 5, and 10 shards? [2 mark]

ANSWER

The following commands return the needed answers:

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

2 shards: shard 1, 5 shards: shard 1, 10 shards: shard 5

c) **[2 marks]** 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`.

i. Retrieve the document with `user_id: 55555`. [1 mark]

Model Solution

ANSWER

```
> use mydb
switched to db mydb
> db.user.find({user_id: 55555})
{ "_id" : ObjectId("54ffab111ab929acfb275a1c"), "user_id" :
  55555, "name" : "Matt", "number" : 6778 }
```

ii. Retrieve the document with `user_id: 1`. [1 mark]

ANSWER

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

d) **[2 marks]** Use the configuration with 10 shards. Connect now to the `shell` of the `mongos` process.

i. Retrieve the document with `user_id: 55555`. [1 marks]

ANSWER

```
mongos > use mydb
switched to db mydb
mongos > db.user.find({user_id: 55555})
{ "_id" : ObjectId("54ffab111ab929acfb275a1c"), "user_id" :
  55555, "name" : "Matt", "number" : 6778 }
```

ii. Retrieve the document with `user_id: 1`. [1 mark]

ANSWER

```
mongos> db.user.find({user_id: 1})
{ "_id" : ObjectId("54ffab0d1ab929acfb26811a"), "user_id" :
1, "name" : "Kristina", "number" : 3679 }
```

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

ANSWER

In question c), we connected to the mongo shell of a `mongod` server. The `mongod` server can access its shard only, since it can't contact `config` servers to get routing information. The mongos process is the router. It connects to a `config` server to get routing information.

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

i. Retrieve the document with `user_id: 55555`. [2 marks]

ANSWER

```
mongos > use mydb
switched to db mydb
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`. [2 marks]

ANSWER

```
mongos> db.user.find({user_id: 1})
{ "_id" : ObjectId("54ffab0d1ab929acfb26811a"), "user_id" :
1, "name" : "Kristina", "number" : 3679 }
```

iii. What percentage (roughly) of your database became unavailable? Will it become available again if you restart the `mongod` server? [4 marks]

ANSWER

Roughly 10%
Yes.

Model Solution

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()`.

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.

ANSWER

The master's port is 27020

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

i. Retrieve the document having `user_id: 1` from the `mydb.user` collection. [1 mark]

ANSWER

```
embassy: [~] % sharep-mongo connect 0 0
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27020/test
rs0:PRIMARY> use mydb
switched to db mydb
rs0:PRIMARY> db.user.find({user_id: 1})
{ "_id" : ObjectId("5502384c9f32951312291903"), "user_id" :
1, "name" : "Steve", "number" : 3029 }
```

ii. Insert the document

`{"user_id": 100000, "name": "Steve", "number": 0}`

into the `mydb.user` collection. [1 mark]

ANSWER

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

Model Solution

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

i. Retrieve the document having `user_id: 1` from the `mydb.user` collection. [1 mark]

ANSWER

```
embassy: [~] % sharep-mongo connect 0 1
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27021/test
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]

ANSWER

```
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`.

i. View the status of the replica set `rs0` and describe it briefly. [1 mark]

ANSWER

```
embassy: [~] % sharep-mongo stop 0 0
Stopping mongod --port 27020 sharep-rs0-0
```

The server 27020 is down, servers 27021 and 27022 are running. The server 27022 has been elected as a new master.

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]

ANSWER

```
mongos> use mydb
switched to db mydb
mongos> db.user.find({user_id: 1})
{ "_id" : ObjectId("5502384c9f32951312291903"), "user_id" :
  1, "name" : "Steve", "number" : 3029 }
mongos> db.user.insert({"user_id": 100001, "name": "Steve",
  "number": 1})
WriteResult({ "nInserted" : 1 })
```

e) [2 mark] Stop the remaining slave server of the replica set `rs0`.

i. View the status of the replica set `rs0` and describe it briefly.

[1 mark]

ANSWER

```
embassy: [~] % sharep-mongo stop 0 1
Stopping mongod --port 27021 sharep-rs0-1
embassy: [~] % sharep-mongo connect 0 2
MongoDB shell version: 2.6.7
connecting to: 127.0.0.1:27022/test
rs0:SECONDARY> rs.status()
```

The servers 27020 and 27021 are down, server 27022 is running, but became a secondary.

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

ANSWER

```
embassy: [~] % 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"
}
```

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

ANSWER

Subquestions b) and c): A master server can read from and write into its shard, whereas a slave server can't write and (by default) can't read, (although, there is an option to configure a slave for reading to achieve work load balancing).

Subquestion d): If a master server is down, one of the slave servers is elected to undertake the master role. The replica set continues functioning (reading and writing).

Subquestion e): In the case of a three server replica set, if a node is already down and the remaining slave server also gets down, the remaining master server turns to a slave and the whole replica set becomes inaccessible by default.

Submission Instruction:

Submit your answers to assignment questions via the school electronic submission system and hand-in a printed version in the hand-in box on the second floor of the Cotton Building.

Please do not submit any .odt, .zip, or similar files. Also, do not submit your files in toll directory trees. All files in the same directory is just fine.

Additionally, submit your commands for questions: 1, 2, 3, 4, and 5 as separate .js files.

A Short Instruction for Using MongoDB on ECS Workstations

1. MongoDB Scripts

Before you try to use MongoDB on our school workstations, you have to type

```
[~] % need mongodb
```

This command will allow all what is needed to deploy MongoDB configurations. You may want to insert `need mongodb` in your `.cshrc` file to avoid typing `need mongodb` whenever you log_on.

Our programmer Royce Brown produced the following four scripts for deploying different MongoDB configurations:

- `[~] % single-mongo,`
- `[~] % rep-mongo,`
- `[~] % sha-mongo,`
- `[~] % sharep-mongo.`

You can run these scripts at the command line prompt of your home directory. **Just run them without any parameters to see what each one does.**

In Assignemnt3_17 you will use `single-mongo.` In Assignment4_17 you will use `single-mongo, sha-mongo,` and `sharep-mongo.`

After starting a MongoDB configuration (e.g. `single-mongo start`), you can:

- Import a collection from a file into a database by typing:

```
[~] % mongoimport --db <database_name> --collection
    <collection_name> --file <file_name>
```

Connect to a mongo shell by typing:

```
[~] % mongo
>
```

2. Useful `mongo` shell commands

To get help:

```
> help
```

To see existing databases, while being in a `mongo(s)` shell:

```
> show dbs
```

To use an existing database, or to define a new one:

```
> use <database_name>
```

To see collections in the current database:

```
> show collections
```

To exit from a `mongo(s)` shell:

CTRL/d or `exit`

Note: The default database is `test`. If you do not issue a

```
use <database_name>
```

command, all your commands are going to be executed against the `test` database.

Warning:

- In all deployments the same ports are assigned to servers. After finishing a session you have to **stop** all servers of your deployment to release ports for other uses. Failing to do so, you will make trouble to other people (potentially including yourself) wanting to use the same workstation. Later, if you want to use the same deployment again, you just do `<script_name> start` and your deployment will resume functioning reliably. If you don't plan to use a deployment again, don't forget to do `<script_name> cleanall`.
- You are strongly advised to use MongoDB from school lab workstations. The school does not undertake any guarantees for using MongoDB from school servers. You may install and use MongoDB on your laptop, but the school does not undertake any responsibilities for the results you obtain.