Q1)

```
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" }
```

Q2)

```
db.reserves.aggregate([

   {$group: {_id: "$reserves.sailor.sailorId", sailorId: {$first:"$reserves.sailor.sailorId"}, name: {$first:"$reserves.sailor.name"}, address: {$first:"$reserves.sailor.address"}, no_of_reserves: {$sum: 1}}},

   {$match:{_id: {$exists: true}}},{$sort: {no_of_reserves: -1}},

   {$limit: 1},

   {$project: {_id: 0, sailorId: 1, name: 1, address: 1, no_of_reserves:1}}

])
```

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

Q3)

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

```
{ "total_reserves" : 18 }
```

Q4)

```
db.reserves.aggregate([
  {$match:{"reserves.sailor.sailorId": {$exists: true}}},
  {$group:{_id: "$reserves.sailor.sailorId", no_of_reserves: {$sum: { $cond: [ {$not: ["$reserves.date"]}, 0, 1 ] }}}},
  {$group:{_id: null, no_of_sailors: {$sum: 1},total_reserves: {$sum: "$no_of_reserves"}}},
  {$project:{_id: 0, avg_reserves_by_sailors: {$divide: ["$total_reserves", "$no_of_sailors"]}}}
 ])
```

```
{ "avg_reserves_by_sailors" : 2.5714285714285716 }
```

Q5)

```
var sailor_name = "Paul";
var sailor_skills = db.reserves.distinct('reserves.sailor.skills', { 'reserves.sailor.name': sailor_name });
var boat_names = db.reserves.aggregate([
   {$match:{"reserves.boat.driven_by": {$exists: true, $ne: []}}},
   {$group: {_id: "$reserves.boat.name", driven_by: {$first:"$reserves.boat.driven_by"}}},
   {$project: {_id: 0, name: "$_id" ,sailor_can_drive: { $setIsSubset: ['$driven_by', sailor_skills] }}},
   {$match:{"sailor_can_drive": true}},
   {$group:{_id: null, boat_names: { $addToSet: "$name" }}},
   {$project: {_id: 0, boat_names: 1 }}
   ]);
if (boat_names.hasNext()) {
   print(tojson(boat_names.next().boat_names));
}
```

```
[ "Killer Whale", "Penguin", "Night Breeze", "Sea Gull" ]
```

Bonus)

```
var avg_no_res = db.reserves.aggregate([
  {$match:{"reserves.sailor.sailorId": {$exists: true}}},
  {$group:{_id: "$reserves.sailor.sailorId", no_of_reserves: {$sum: { $cond: [ {$not: ["$reserves.date"]}, 0, 1 ] }}}},
  {$group:{_id: null, no_of_sailors: {$sum: 1},total_reserves: {$sum: "$no_of_reserves"}}},
  {$project:{_id: 0, avg_reserves_by_sailors: {$divide: ["$total_reserves", "$no_of_sailors"]}}}
 ]).next().avg_reserves_by_sailors;


db.reserves.aggregate([
    {$match:{"reserves.sailor.sailorId": {$exists: true}}},
    {$group: {_id: "$reserves.sailor.sailorId", name: {$first:"$reserves.sailor.name"}, no_of_reserves: {$sum: 1}}},
    {$match:{no_of_reserves: {$gt: avg_no_res } }}
]);
```

```
{ "_id" : 111, "name" : "Peter", "no_of_reserves" : 3 }
{ "_id" : 818, "name" : "Milan", "no_of_reserves" : 6 }
{ "_id" : 707, "name" : "James", "no_of_reserves" : 3 }
```

Q6)-a)

Use "**shardCollection"** command based on ranges partitioning.

Q6)-b)-i

Using **"db.user.getShardDistribution()**" command can get the following result.

2 shards case: 6 in each shard;

5 shards case: 2 or 3 in each shard;

10 shards case: 1 or 2 in each shard;

Estimating formula:

The number of chunks = the number of documents * the average size of document /the number of shards / the size of chunks

If average document size is 100B, then 100000 documents' total size is

100000*100B/1024/1024=10MB.

In 2 shards case, each shard has 10MB/2=5MB size documents, if chunk's size is 1MB, there will be 5 chunks in each shard.

In 5 shards, each shard has 10MB/5=2MB size documents, if chunk's size is 1MB, there will be 2 chunks in each shard.

In 10 shards, each shard has 10MB/10=1MB size documents, if chunk's size is 1MB, there will be 1 chunks in each shard.

Q6)-b)-ii

Using **" db.printShardingStatus(true)**" command can get the following result.

2 shards case: shard0001

5 shards case: shard0001

10 shards case: shard0005

Q6)-c)-i

db.user.find({"user_id":55555});

```
{ "_id" : ObjectId("59238f4e6fe2b378a74227a9"), "user_id" : 55555, "name" : "Jeff", "number" : 2411 }
{ "_id" : ObjectId("59238aebec44dca1d32426f1"), "user_id" : 55555, "name" : "George", "number" : 1429 }
{ "_id" : ObjectId("5923908f364c7e908668064e"), "user_id" : 55555, "name" : "Jeff", "number" : 2208 }
```

Q6)-c)-ii

db.user.find({"user_id":1});

```
No results
```

Q6)-d)-i

db.user.find({"user_id":55555});

```
{ "_id" : ObjectId("59238f4e6fe2b378a74227a9"), "user_id" : 55555, "name" : "Jeff", "number" : 2411 }
{ "_id" : ObjectId("59238aebec44dca1d32426f1"), "user_id" : 55555, "name" : "George", "number" : 1429 }
{ "_id" : ObjectId("5923908f364c7e908668064e"), "user_id" : 55555, "name" : "Jeff", "number" : 2208 }
```

Q6)-d)-ii

db.user.find({"user_id":1});

```
{ "_id" : ObjectId("59238aeaec44dca1d3234def"), "user_id" : 1, "name" : "Tracy", "number" : 282 }
{ "_id" : ObjectId("59238f4d6fe2b378a7414ea7"), "user_id" : 1, "name" : "Eliot", "number" : 9898 }
{ "_id" : ObjectId("5923908e364c7e9086672d4c"), "user_id" : 1, "name" : "Tracy", "number" : 9979 }
```

Q6)-e)

A shard contains a subset of sharded data for a sharded cluster.Together, the cluster's shards hold the entire data set for the cluster.

In Q6)-c), performing queries on a single shard only returns a subset of data, cannot returns other shards data.

In Q6)-d), connecting to the mongos to perform cluster level operations returns the entire data set for the cluster.

Q6)-f)-i

sha-mongo stop 5

```
Stopping mongod --port 27025 shadb5
```

db.user.find ({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"
}
```

Q6)-f)-ii

db.user.find ({user_id:1});

```
{ "_id" : ObjectId("59261be36ac2019c74102f01"), "user_id" : 1, "name" : "Katie", "number" : 3962 }
```

Q6)-f)-iii

Roughly 10% became unavailable.

restart the mongod server using command "**sha-mongo start 5**", then data became available.

```
mongos> db.user.find ({user_id:55555});
{ "_id" : ObjectId("59261be56ac2019c74110803"), "user_id" : 55555, "name" : "George", "number" : 3004 }
mongos> db.user.find ({user_id:1});
{ "_id" : ObjectId("59261be36ac2019c74102f01"), "user_id" : 1, "name" : "Katie", "number" : 3962 }
```

Q7)-a)

rs.status();

```
> rs.status()
{
	"set" : "rs0",
	"date" : ISODate("2017-05-25T00:29:35Z"),
	"myState" : 1,
	"members" : [
		{
			"_id" : 0,
			"name" : "regent.ecs.vuw.ac.nz:27020",
			"health" : 1,
			"state" : 1,
			"stateStr" : "PRIMARY",
			"uptime" : 256,
			"optime" : Timestamp(1495672147, 1),
			"optimeDate" : ISODate("2017-05-25T00:29:07Z"),
			"electionTime" : Timestamp(1495672148, 1),
			"electionDate" : ISODate("2017-05-25T00:29:08Z"),
			"self" : true
		}
	],
	"ok" : 1
}
```

27020 is the port number.

Q7)-b)-i

sharep-mongo connect 0 0

use mydb;

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

Q7)-b)-ii

db.user.insert({"user_id": 100000, "name": "Steve", "number": 0});

Q7)-C)-i

sharep-mongo connect 0 1

use mydb;

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

Q7)-c)-ii

db.user.insert({"user_id": 100000, "name": "Steve", "number": 0});

Q7)-d)-i

sharep-mongo stop 0 0

rs.status()

If the primary becomes inaccessible, the remaining members of the replica set have to elect a new primary by voting.

desicible it briefly

Q7)-d)-ii

sharep-mongo connect ;

use mydb;

db.user.find({user_id:1});

db.user.insert({"user_id": 100000, "name": "Steve", "number": 0});

Q7)-e)-i

sharep-mongo stop 0 2 or 1 (stop remaining slave server)

sharep-mongo connect

rs.status();

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.

Q7)-e)-ii

sharep-mongo connect;

use mydb;

db.user.find({user_id:1});

Q7)-f)

If a sencondary becomes inaccessible, the replica set can continue to function without it.

if there is no master in the Replica Set. The eventually consistency cannot be meet thus the data in that set is not available for reading.