Practical No: 01

Aim: Setting up and Exploring MongoDB.
a) Install MongoDB on your local machine or lab server.
b) Create a new MongoDB database and collection.
c) Insert sample data into the collection.
d) Retrieve and display data from the collection using MongoDB queries.

1. Create a database name userdb.
use userdb;

```
>_MONGOSH

> use userdb;
< switched to db userdb
userdb>
```

2. Creating a New Collection:
db.createCollection("users")

```
> db.createCollection("users")
< { ok: 1 }
userdb>
```

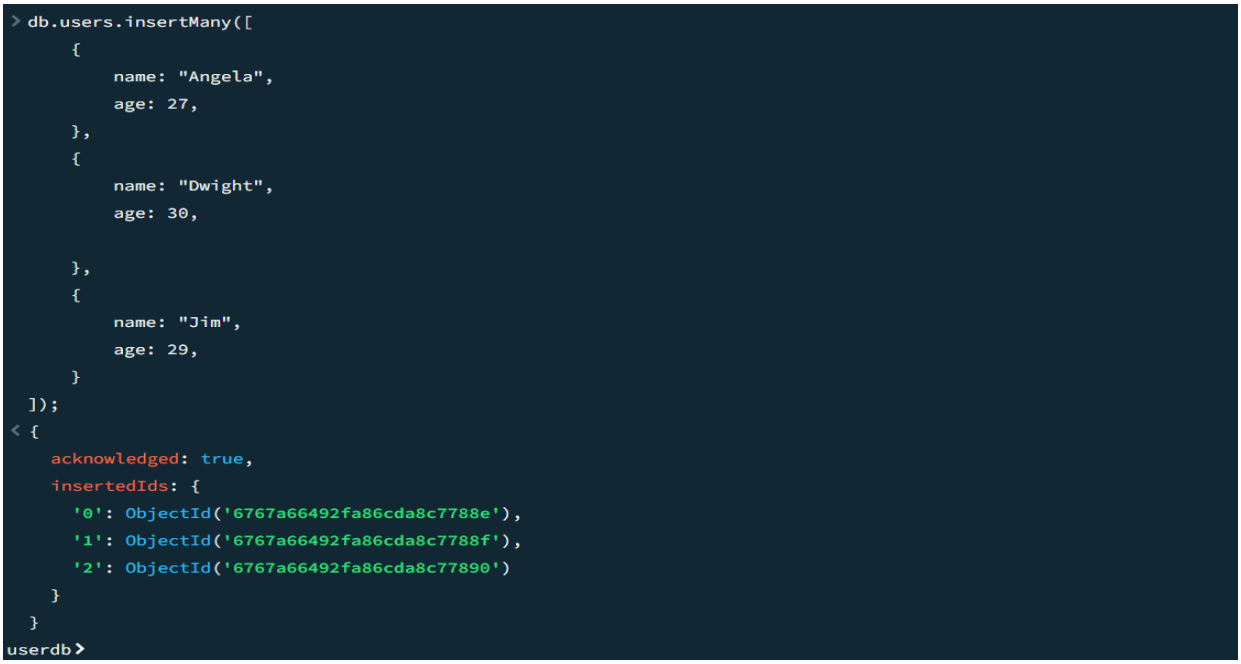
3. insertOne():
db.users.insertOne({
  name: "Angela",
  age: 27,
});

```
> db.users.insertOne({
    name: "Angela",
    age: 27,
  });
< {
    acknowledged: true,
    insertedId: ObjectId('6767a57892fa86cda8c7788d')
  }
userdb>
```

4. insertMany():

```
db.users.insertMany([
  {
    name: "Angela",
    age: 27,
  },
  {
    name: "Dwight",
    age: 30,

  },
  {
    name: "Jim",
    age: 29,
  }
]);
```



5. find():
db.users.find()

```
> db.users.find()
< {
    _id: ObjectId('6767a57892fa86cda8c7788d'),
    name: 'Angela',
    age: 27
  }
  {
    _id: ObjectId('6767a66492fa86cda8c7788e'),
    name: 'Angela',
    age: 27
  }
  {
    _id: ObjectId('6767a66492fa86cda8c7788f'),
    name: 'Dwight',
    age: 30
  }
  {
    _id: ObjectId('6767a66492fa86cda8c77890'),
    name: 'Jim',
    age: 29
  }
userdb>
```

6. findOne():
db.users.findOne({ name: "Jim" })

```
> db.users.findOne({ name: "Jim" })
< {
    _id: ObjectId('6767a66492fa86cda8c77890'),
    name: 'Jim',
    age: 29
  }
userdb>
```

7. updateOne():
db.users.updateOne({ name: "Angela" }, { $set: { email: "angela@gmail.com" } })

```
> db.users.updateOne({ name: "Angela" }, { $set: { email: "angela@gmail.com" } })
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
userdb>
```

db.users.find()

```
> db.users.find()
< {
    _id: ObjectId('6767a57892fa86cda8c7788d'),
    name: 'Angela',
    age: 27,
    email: 'angela@gmail.com'
  }
  {
    _id: ObjectId('6767a66492fa86cda8c7788e'),
    name: 'Angela',
    age: 27
  }
  {
    _id: ObjectId('6767a66492fa86cda8c7788f'),
    name: 'Dwight',
    age: 30
  }
  {
    _id: ObjectId('6767a66492fa86cda8c77890'),
    name: 'Jim',
    age: 29
  }
userdb>
```

8. updateMany():
db.users.updateMany({ age: { $lt: 30 } }, { $set: { status: "active" } })

```
> db.users.updateMany({ age: { $lt: 30 } }, { $set: { status: "active" } })
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 3,
    modifiedCount: 3,
    upsertedCount: 0
  }
userdb>
```

db.users.find()

```
> db.users.find()
< {
    _id: ObjectId('6767a57892fa86cda8c7788d'),
    name: 'Angela',
    age: 27,
    email: 'angela@gmail.com',
    status: 'active'
  }
  {
    _id: ObjectId('6767a66492fa86cda8c7788e'),
    name: 'Angela',
    age: 27,
    status: 'active'
  }
  {
    _id: ObjectId('6767a66492fa86cda8c7788f'),
    name: 'Dwight',
    age: 30
  }
  {
    _id: ObjectId('6767a66492fa86cda8c77890'),
    name: 'Jim',
    age: 29,
    status: 'active'
  }
userdb>
```

9. deleteOne():
db.users.deleteOne({ name: "Angela" })

```
> db.users.deleteOne({ name: "Angela" })
< {
    acknowledged: true,
    deletedCount: 1
  }
userdb>
```

db.users.find()

```
> db.users.find()
< {
    _id: ObjectId('6767a66492fa86cda8c7788e'),
    name: 'Angela',
    age: 27,
    status: 'active'
  }
  {
    _id: ObjectId('6767a66492fa86cda8c7788f'),
    name: 'Dwight',
    age: 30
  }
  {
    _id: ObjectId('6767a66492fa86cda8c77890'),
    name: 'Jim',
    age: 29,
    status: 'active'
  }
userdb>
```

10. deleteMany():
db.users.deleteMany({ age: { $lt: 30 } })

```
> db.users.deleteMany({ age: { $lt: 30 } })
< {
    acknowledged: true,
    deletedCount: 2
  }
userdb>
```

db.users.find()

```
> db.users.find()
< {
    _id: ObjectId('6767a66492fa86cda8c7788f'),
    name: 'Dwight',
    age: 30
  }
userdb>
```

11. drop():
db.users.drop()

```
> db.users.drop()
< true
> db.users.find()
<
userdb>
```

Practical No: 02

Aim: Interacting with Redis.
a) Install Redis on your lab server or local machine.
b) Store and retrieve data in Redis using various data structures like strings, lists, and sets.
c) Implement basic Redis commands for data manipulation and retrieval.

Setting a value can be done with the set command:
1. localhost:6379> set key:1 value1new

```
> set key:1 value1new
"OK"

>
```

2. localhost:6379> get key:1

```
> get key:1
"value1new"

>
```

Creating a value can also be done in the same way:
3. localhost:6379> get key:3

```
> get key:3
(nil)

>
```

4. localhost:6379> set key:3 value3new

```
> set key:3 value3new
"OK"

>
```

5. localhost:6379> get key:3

```
> get key:3
"value3new"

>
```

And deletion can be done with the del command:
6. localhost:6379> del key:3

```
> del key:3
(integer) 1

>
```

7. localhost:6379> get key:3

```
>  get key:3
(nil)

>
```

Practical No: 03

Aim: Working with HBase.
a) Set up an HBase cluster in a lab environment.
b) Create an HBase table and define column families.
c) Insert sample data into the table.
d) Perform CRUD operations and retrieval of data in HBase.

1. Create table
create 'emp', 'personal data', 'professional data'

```
hbase(main):001:0> create 'emp', 'personal data', 'professional data'
0 row(s) in 1.5060 seconds

=> Hbase::Table - emp
hbase(main):002:0>
```

2. Insert data

put 'emp','1','personal data:name','raju'
put 'emp','1','personal data:city','hyderabad'
put 'emp','1','professional data:designation','manager'
put 'emp','1','professional data:salary','50000'

```
hbase(main):002:0> put 'emp','1','personal data:name','raju'
0 row(s) in 0.2010 seconds

hbase(main):003:0> put 'emp','1','personal data:city','hyderabad'
0 row(s) in 0.0170 seconds

hbase(main):004:0> put 'emp','1','professional data:designation','manager'
0 row(s) in 0.0470 seconds

hbase(main):005:0> put 'emp','1','professional data:salary','50000'
0 row(s) in 0.0130 seconds

hbase(main):006:0>
```

3. Select Data

scan 'emp'

```
hbase(main):006:0> scan 'emp'
ROW                                     COLUMN+CELL
 1                                      column=personal data:city, timestamp=1736057161605, value=hyderabad
 1                                      column=personal data:name, timestamp=1736057161487, value=raju
 1                                      column=professional data:designation, timestamp=1736057161685, value=manager
 1                                      column=professional data:salary, timestamp=1736057161880, value=50000
1 row(s) in 0.0630 seconds

hbase(main):007:0>
```

4. Update Data

put 'emp','1','personal:city','Delhi'

```
hbase(main):001:0> put 'emp','row1','personal data:city','Delhi'
0 row(s) in 0.3520 seconds

hbase(main):002:0>
```

5. Select Data

scan 'emp'

```
hbase(main):002:0> scan 'emp'
ROW                                     COLUMN+CELL
 1                                      column=personal data:city, timestamp=1736058512530, value=delhi
 1                                      column=personal data:name, timestamp=1736058396364, value=ajay
 1                                      column=professional data:designation, timestamp=1736057161685, value=manager
 1                                      column=professional data:salary, timestamp=1736057161880, value=50000
 row1                                   column=personal data:city, timestamp=1736058977276, value=Delhi
2 row(s) in 0.0490 seconds

hbase(main):003:0>
```

6. Retrieve Data

get 'emp', 'row1', {COLUMN ⇒ 'personal data:name'}

```
hbase(main):005:0> get 'emp','row1',{COLUMN => 'personal data:name'}
COLUMN                  CELL
0 row(s) in 0.0170 seconds

hbase(main):006:0>
```

7. Delete Data

delete 'emp', '1', 'personal data:city'

```
hbase(main):006:0> delete 'emp', '1', 'personal data:city'
0 row(s) in 0.0820 seconds

hbase(main):007:0> scan 'emp'
ROW                                   COLUMN+CELL
 1                                    column=personal data:name, timestamp=1736058396364, value=ajay
 1                                    column=professional data:designation, timestamp=1736057161685, value=manager
 1                                    column=professional data:salary, timestamp=1736057161880, value=50000
 row1                                 column=personal data:city, timestamp=1736058977276, value=Delhi
2 row(s) in 0.0510 seconds

hbase(main):008:0>
```

deleteall 'emp','1'

```
hbase(main):008:0> deleteall 'emp','1'
0 row(s) in 0.0280 seconds

hbase(main):009:0> scan 'emp'
ROW                                   COLUMN+CELL
 row1                                 column=personal data:city, timestamp=1736058977276, value=Delhi
1 row(s) in 0.0210 seconds

hbase(main):010:0>
```

Practical No: 04

Aim: Apache Cassandra Operations.
a) Install and configure Apache Cassandra in a lab environment.
b) Create a keyspace and define a table schema.
c) Insert data into the table.
d) Perform CRUD operations and query data from Apache Cassandra.

1. check Databases:
describe keyspaces

```
onnected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 2.2.3 | CQL spec 3.3.1 | Native protocol v4]
Use HELP for help.
WARNING: pyreadline dependency missing.  Install to enable tab completion.
cqlsh> describe keyspaces

system_traces  system_auth  system  "OpsCenter"  system_distributed
```

2. Create Keyspaces:
CREATE KEYSPACE Employee_Data
... WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 3};

```
cqlsh> CREATE KEYSPACE Employee_Data
   ... WITH replication = {'class':'SimpleStrategy','replication_factor':3};
```

3. Use Keyspaces:
use Employee_Data;

```
cqlsh> use Employee_Data;
```

4. Create table emp:
CREATE TABLE emp(
emp_id int PRIMARY KEY,
emp_name text,
emp_city text,
emp_sal varint,
emp_phone varint
);

```
cqlsh> use Employee_Data;
cqlsh:employee_data> CREATE TABLE emp(
                 ... emp_id int PRIMARY KEY,
                 ... emp_name text,
                 ... emp_city text,
                 ... emp_sal varint,
                 ... emp_phone varint
                 ... );
```

5. Verification :
select * from emp;

```
                ... );
cqlsh:employee_data> select*from emp;

 emp_id | emp_city | emp_name | emp_phone | emp_sal
--------+----------+----------+-----------+---------
```

6. Creating Data in a Table:
cqlsh:employee_data> INSERT INTO emp (emp_id, emp_name, emp_city,

... emp_phone, emp_sal) VALUES(1,'ram', 'Hyderabad', 9848022338, 50000);

cqlsh:employee_data> INSERT INTO emp (emp_id, emp_name, emp_city,

... emp_phone, emp_sal) VALUES(2,'robin', 'Hyderabad', 9848022339, 40000);

cqlsh:employee_data> INSERT INTO emp (emp_id, emp_name, emp_city,

... emp_phone, emp_sal) VALUES(3,'rahman', 'Chennai', 9848022330, 45000);

```
qlsh:employee_data> INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal)
                ... VALUES(1, 'ram', 'Hyderabad', 9848022338, 50000);
qlsh:employee_data> INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal)
                ... VALUES(2,'robin', 'Hyderabad', 9848022339, 40000);
qlsh:employee_data> INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal)
                ... VALUES(3,'rahman', 'Chennai', 9848022330, 45000);
qlsh:employee_data> select * from emp;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 | Hyderabad |    robin | 9848022339 |   40000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
```

7. verification
select * from emp;

```
qlsh:employee_data> INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal)
                ... VALUES(1, 'ram', 'Hyderabad', 9848022338, 50000);
qlsh:employee_data> INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal)
                ... VALUES(2,'robin', 'Hyderabad', 9848022339, 40000);
qlsh:employee_data> INSERT INTO emp (emp_id, emp_name, emp_city, emp_phone, emp_sal)
                ... VALUES(3,'rahman', 'Chennai', 9848022330, 45000);
qlsh:employee_data> select * from emp;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 | Hyderabad |    robin | 9848022339 |   40000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
```

8. update emp:
cqlsh:employee_data> UPDATE emp SET emp_city='Delhi',emp_sal=50000

... WHERE emp_id=2;
cqlsh:employee_data> select *from emp;

```
cqlsh:employee_data> UPDATE emp SET emp_city='Delhi',emp_sal=50000
                ... WHERE emp_id=2;
cqlsh:employee_data> select *from emp;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
      3 |   Chennai |   rahman | 9848022330 |   45000

(3 rows)
```

9. Reading Data from emp:
SELECT emp_name, emp_sal from emp;

```
(3 rows)
cqlsh:employee_data> SELECT emp_name, emp_sal from emp;

 emp_name | emp_sal
----------+---------
      ram |   50000
    robin |   50000
   rahman |   45000
```

10. Delete data from emp:
DELETE emp_sal FROM emp WHERE emp_id=3;

```
(3 rows)
cqlsh:employee_data> DELETE emp_sal FROM emp WHERE emp_id=3;
cqlsh:employee_data> select * from emp;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
      3 |   Chennai |   rahman | 9848022330 |    null

(3 rows)
cqlsh:employee_data>
```

11. Verification:
select * from emp;

```
(3 rows)
cqlsh:employee_data> DELETE emp_sal FROM emp WHERE emp_id=3;
cqlsh:employee_data> select * from emp;

 emp_id | emp_city  | emp_name | emp_phone  | emp_sal
--------+-----------+----------+------------+---------
      1 | Hyderabad |      ram | 9848022338 |   50000
      2 |     Delhi |    robin | 9848022339 |   50000
      3 |   Chennai |   rahman | 9848022330 |    null

(3 rows)
cqlsh:employee_data>
```

Practical No: 05

Aim: Querying MongoDB and HBase.
a) Write and execute MongoDB queries to retrieve specific data from a collection.

```
db.inventory.insertMany( [
  { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
  { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
  { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
  { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
  { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

db.inventory.find({})

The following example selects from the inventory collection all documents where the status equals "D":

db.inventory.find({ status: "D" })

```
> db.inventory.find( { status: "D" } )
< {
    _id: ObjectId('6768e79b3239b4ddf6b99a22'),
    item: 'paper',
    qty: 100,
    size: {
      h: 8.5,
      w: 11,
      uom: 'in'
    },
    status: 'D'
  }
  {
    _id: ObjectId('6768e79b3239b4ddf6b99a23'),
    item: 'planner',
    qty: 75,
    size: {
      h: 22.85,
      w: 30,
      uom: 'cm'
    },
    status: 'D'
  }
test >
```

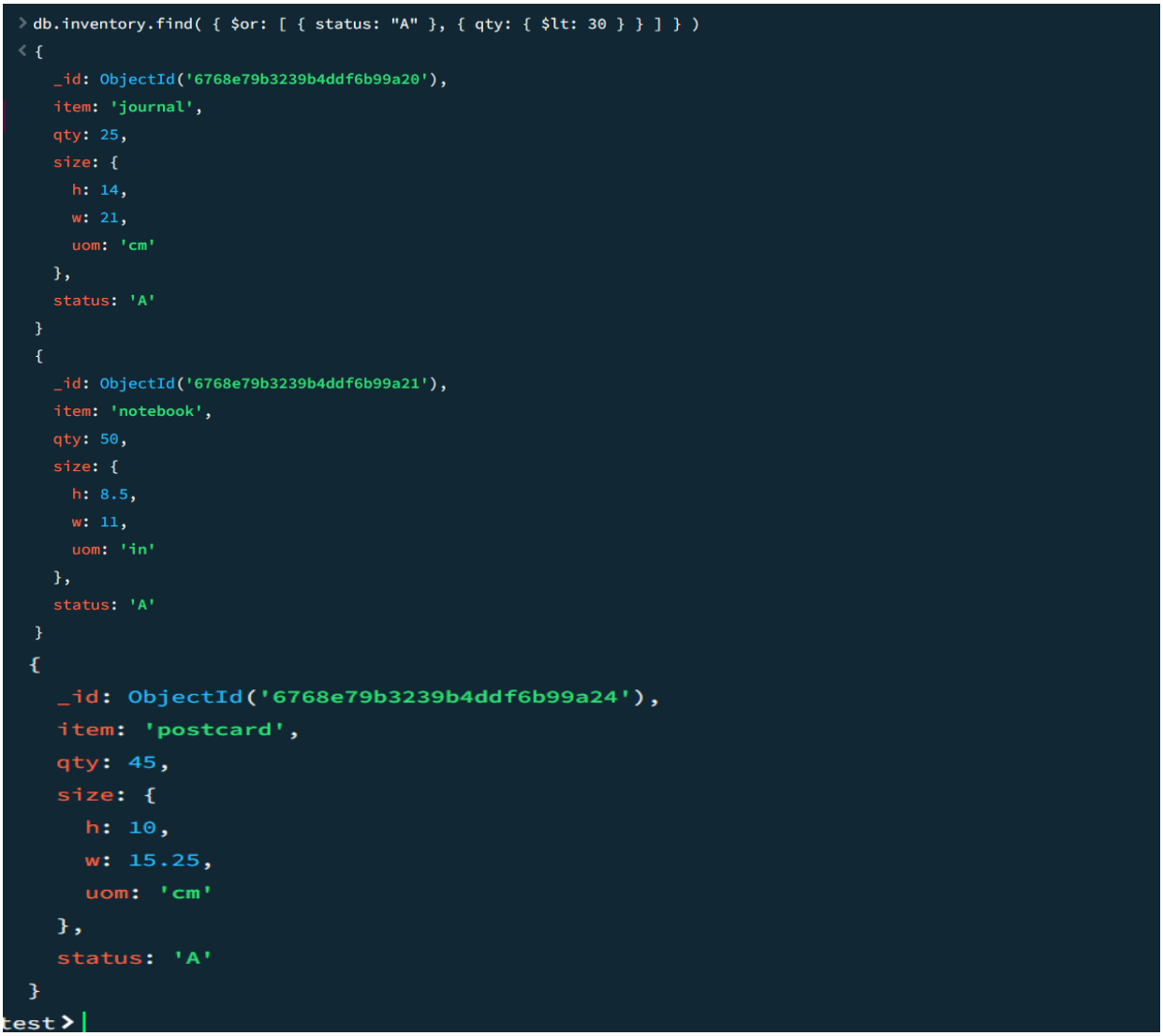
The following example retrieves all documents in the inventory collection where the status equals "A" and qty is less than ($lt) 30:

db.inventory.find({ status: "A", qty: { $lt: 30 } })

```
> db.inventory.find( { status: "A", qty: { $lt: 30 } } )
< {
    _id: ObjectId('6768e79b3239b4ddf6b99a20'),
    item: 'journal',
    qty: 25,
    size: {
      h: 14,
      w: 21,
      uom: 'cm'
    },
    status: 'A'
  }
test >
```

The following example retrieves all documents in the inventory collection where the status equals "A" or qty is less than ($lt) 30:

db.inventory.find({ $or: [{ status: "A" }, { qty: { $lt: 30 } }] })



In the following example, the compound query document selects all documents in the collection where the status equals "A" and either qty is less than ($lt) 30 or item starts with the character p:

```
db.inventory.find( {
    status: "A",
    $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
} )
```

```
> db.inventory.find( {
        status: "A",
        $or: [ { qty: { $lt: 30 } }, { item: /^p/ } ]
  } )
< {
    _id: ObjectId('6768e79b3239b4ddf6b99a20'),
    item: 'journal',
    qty: 25,
    size: {
      h: 14,
      w: 21,
      uom: 'cm'
    },
    status: 'A'
  }
  {
    _id: ObjectId('6768e79b3239b4ddf6b99a24'),
    item: 'postcard',
    qty: 45,
    size: {
      h: 10,
      w: 15.25,
      uom: 'cm'
    },
    status: 'A'
  }
test >
```

Practical No: 06

Aim: Redis Data Manipulation**.**
a) Use Redis commands to manipulate and modify data stored in different data structures.
b) Retrieve specific data using Redis query operations.

- LPUSH adds a new element to the head of a list;
- RPUSH adds to the tail.
- LPOP removes and returns an element from the head of a list;
- RPOP does the same but from the tails of a list.
- LLEN returns the length of a list.
- LTRIM reduces a list to the specified range of elements.
- Treat a list like a queue (first in, first out):

>_ Redis CLI

LPUSH bikes:repairs bike:1

```
> LPUSH bikes:repairs bike:1
(integer) 1

>
```

LPUSH bikes:repairs bike:2

```
> LPUSH bikes:repairs bike:2
(integer) 2

>
```

RPOP bikes:repairs

```
> RPOP bikes:repairs
"bike:1"

>
```

RPOP bikes:repairs

```
> RPOP bikes:repairs
"bike:2"

>
```

>_ Redis CLI

LPUSH bikes:repairs bike:1

```
> LPUSH bikes:repairs bike:1
(integer) 1

>
```

LPUSH bikes:repairs bike:2

```
> LPUSH bikes:repairs bike:2
(integer) 2

>
```

LPOP bikes:repairs

```
> LPOP bikes:repairs
"bike:2"

>
```

LPOP bikes:repairs

```
> LPOP bikes:repairs
"bike:1"

>
```

>_ Redis CLI

LLEN bikes:repairs

```
> LLEN bikes:repairs
(integer) 0

>
```

>_ Redis CLI

LPUSH bikes:repairs bike:1

```
> LPUSH bikes:repairs bike:1
(integer) 1

>
```

LPUSH bikes:repairs bike:2

```
> LPUSH bikes:repairs bike:2
(integer) 2

>
```

LRANGE bikes:repairs 0 -1

```
> LRANGE bikes:repairs 0 -1
1) "bike:1"

>
```

LRANGE bikes:finished 0 -1

```
> LRANGE bikes:finished 0 -1
(empty list or set)

>
```

>_ Redis CLI

RPUSH bikes:repairs bike:1 bike:2 bike:3 bike:4 bike:5

```
> RPUSH bikes:repairs bike:1 bike:2 bike:3 bike:4 bike:5
(integer) 9
```

LTRIM bikes:repairs 0 2

```
> LTRIM bikes:repairs 0 2
"OK"
```

LRANGE bikes:repairs 0 -1

```
> LRANGE bikes:repairs 0 -1
1) "bike:2"
2) "bike:2"
3) "bike:2"

>
```

Practical No: 07

Aim: Implementing Indexing in MongoDB.
a) Create an index on a specific field in a MongoDB collection.
b) Measure the impact of indexing on query performance.

1. Create Indexes:
db.products.createIndex(
{ item: 1, quantity: -1 } ,
{ name: "query for inventory" }
)

```
> db.products.createIndex(
    { item: 1, quantity: -1 } ,
    { name: "query for inventory" }
  )
< query for inventory
test>
```

2. Get Indexes:
db.products.getIndexes()

```
> db.products.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    { v: 2, key: { item: 1, quantity: -1 }, name: 'query for inventory' }
  ]
test>
```

3. Drop Indexes:
db.products.dropIndexes()

```
> db.products.dropIndexes()
< {
    nIndexesWas: 2,
    msg: 'non-_id indexes dropped for collection',
    ok: 1
  }
test>
```

db.products.getIndexes()

```
> db.products.getIndexes()
< [ { v: 2, key: { _id: 1 }, name: '_id_' } ]
test>
```

Practical No: 08

Aim: Data Storage in Redis.
a) Implement caching functionality using Redis as a cache store.
b) Store and retrieve data from Redis cache using appropriate commands.

- SADD adds a new member to a set.
- SREM removes the specified member from the set.
- SISMEMBER tests a string for set membership.
- SINTER returns the set of members that two or more sets have in common (i.e., the intersection).
- SCARD returns the size (a.k.a. cardinality) of a set.

- Store the sets of bikes racing in France and the USA. Note that if you add a member that already exists, it will be ignored.

>_ Redis CLI

SADD bikes:racing:france bike:1

```
> SADD bikes:racing:france bike:1
(integer) 1
```

SADD bikes:racing:france bike:1

```
> SADD bikes:racing:france bike:1
(integer) 0

>
```

SADD bikes:racing:france bike:2 bike:3

```
> SADD bikes:racing:france bike:2 bike:3
(integer) 2

>
```

SADD bikes:racing:usa bike:1 bike:4

```
> SADD bikes:racing:usa bike:1 bike:4
(integer) 2

>
```

- **Check whether bike: 1 or bike:2 are racing in the US.**

>_ Redis CLI

SISMEMBER bikes:racing:usa bike:1

```
> SISMEMBER bikes:racing:usa bike:1
(integer) 1

>
```

SISMEMBER bikes:racing:usa bike:2

```
> SISMEMBER bikes:racing:usa bike:2
(integer) 0

>
```

- **Which bikes are competing in both races?**

>_ Redis CLI

SINTER bikes:racing:france bikes:racing:usa

```
> SINTER bikes:racing:france bikes:racing:usa
1) "bike:1"

>
```

- **How many bikes are racing in France?**

>_ Redis CLI

SCARD bikes:racing:france

```
> SCARD bikes:racing:france
(integer) 3

>
```