ASSIGNMENT NO :16

Create a collection named Book. (book_isbn,title,punlisher_name,author(Name, Address, Phone No[landline, mobile]), publisher_city, price,copies)
i. a. Add 5 documents in the collection with keys
b. Give details of Books whose Publisher lives in “Pune”.
c. Delete name Book from Book whose name start with “D”
d. Change the city of publisher “Pearson” to “Pune”.
e. Find the details of publisher named “Pearson”.

1)create database

```
test> use exam
switched to db exam
```

2)create collection

```
exam> db.createCollection("Book")
{ ok: 1 }
```

a) Add 5 documents in the collection with keys.

```
exam>
db.Book.insertOne({"book_isbn":101,"title":"dbms","publisher_name":"ram","author":{"name":"a
uthor1","address":"pune","phone_no":{"landline":"123-456-7890","mobile":"1234567890"}},"publi
sher_city":"pune","price":99,"copies":10})
{
  acknowledged: true,
  insertedId: ObjectId("6558289d3b4c39fe061569f6")
}
exam>
db.Book.insertOne({"book_isbn":102,"title":"toc","publisher_name":"sham","author":{"name":"aut
hor2","address":"nashik","phone_no":{"landline":"123-456-3453","mobile":"7080908967"}},"publi
sher_city":"baramati","price":70,"copies":20})
{
  acknowledged: true,
  insertedId: ObjectId("6558290e3b4c39fe061569f7")
}
exam>
db.Book.insertOne({"book_isbn":103,"title":"iot","publisher_name":"shubham","author":{"name":"
author3","address":"dhule","phone_no":{"landline":"345-456-3453","mobile":"9191908967"}},"pu
blisher_city":"malegaon","price":170,"copies":40})
{
  acknowledged: true,
  insertedId: ObjectId("6558296e3b4c39fe061569f8")
}
exam>
db.Book.insertOne({"book_isbn":104,"title":"oop","publisher_name":"aniket","author":{"name":"au
thor4","address":"supe","phone_no":{"landline":"345-546-3453","mobile":"9100008967"}},"publis
her_city":"nagar","price":190,"copies":80})
{
  acknowledged: true,
  insertedId: ObjectId("655829da3b4c39fe061569f9")
}
exam>
db.Book.insertOne({"book_isbn":105,"title":"python","publisher_name":"shivtej","author":{"name":
"author5","address":"saswad","phone_no":{"landline":"945-546-3453","mobile":"9098976756"}},"
publisher_city":"balewadi","price":290,"copies":110})
{
  acknowledged: true,
  insertedId: ObjectId("65582a343b4c39fe061569fa")
}
```

exam> db.Book.find()

```
[
   {
    _id: ObjectId("6558306c3b4c39fe061569fb"),
    book_isbn: 101,
    title: 'dbms',
    publisher_name: 'ram',
    author: {
      name: 'author1',
      address: 'pune',
      phone_no: { landline: '123-456-7890', mobile: '1234567890' }
    },
    publisher_city: 'pune',
    price: 99,
    copies: 10
  }

  {
    _id: ObjectId("6558290e3b4c39fe061569f7"),
    book_isbn: 102,
    title: 'toc',
    publisher_name: 'sham',
    author: {
      name: 'author2',
      address: 'nashik',
      phone_no: { landline: '123-456-3453', mobile: '7080908967' }
    },
    publisher_city: 'baramati',
    price: 70,
    copies: 20
  },
  {
    _id: ObjectId("6558296e3b4c39fe061569f8"),
    book_isbn: 103,
    title: 'iot',
    publisher_name: 'shubham',
    author: {
      name: 'author3',
      address: 'dhule',
      phone_no: { landline: '345-456-3453', mobile: '9191908967' }
    },
    publisher_city: 'pune',
    price: 170,
    copies: 40
  },
  {
    _id: ObjectId("655829da3b4c39fe061569f9"),
    book_isbn: 104,
    title: 'oop',
    publisher_name: 'aniket',
    author: {
      name: 'author4',
      address: 'supe',
      phone_no: { landline: '345-546-3453', mobile: '9100008967' }
    },
    publisher_city: 'nagar',
    price: 190,
    copies: 80
  },
  {
    _id: ObjectId("65582a343b4c39fe061569fa"),
    book_isbn: 105,
    title: 'python',
    publisher_name: 'shivtej',
    author: {
      name: 'author5',
      address: 'saswad',
      phone_no: { landline: '945-546-3453', mobile: '9098976756' }
    },
    publisher_city: 'balewadi',
    price: 290,
    copies: 110
  },
 ]
```

b. Give details of Books whose Publisher lives in “Pune”.

```
exam> db.Book.find({"author.address":{$regex:/pune/i}});
[
  {
    _id: ObjectId("6558289d3b4c39fe061569f6"),
    book_isbn: 101,
    title: 'dbms',
    publisher_name: 'ram',
    author: {
      name: 'author1',
      address: 'pune',
      phone_no: { landline: '123-456-7890', mobile: '1234567890' }
    },
    publisher_city: 'pune',
    price: 99,
    copies: 10
  }
]
```

c. Delete name Book from Book whose name start with “D”

```
exam> db.Book.deleteOne({"title":{$regex:/^d/i}});
{ acknowledged: true, deletedCount: 1 }

exam> db.Book.find()
[
  {
    _id: ObjectId("6558290e3b4c39fe061569f7"),
    book_isbn: 102,
    title: 'toc',
    publisher_name: 'sham',
    author: {
      name: 'author2',
      address: 'nashik',
      phone_no: { landline: '123-456-3453', mobile: '7080908967' }
    },
    publisher_city: 'baramati',
    price: 70,
    copies: 20
  },
  {
    _id: ObjectId("6558296e3b4c39fe061569f8"),
    book_isbn: 103,
    title: 'iot',
    publisher_name: 'shubham',
    author: {
      name: 'author3',
      address: 'dhule',
      phone_no: { landline: '345-456-3453', mobile: '9191908967' }
    },
    publisher_city: 'pune',
    price: 170,
    copies: 40
  },
  {
    _id: ObjectId("655829da3b4c39fe061569f9"),
    book_isbn: 104,
    title: 'oop',
    publisher_name: 'aniket',
    author: {
      name: 'author4',
      address: 'supe',
      phone_no: { landline: '345-546-3453', mobile: '9100008967' }
    },
    publisher_city: 'nagar',
    price: 190,
    copies: 80
  },
  {
    _id: ObjectId("65582a343b4c39fe061569fa"),
    book_isbn: 105,
    title: 'python',
    publisher_name: 'shivtej',
    author: {
      name: 'author5',
      address: 'saswad',
      phone_no: { landline: '945-546-3453', mobile: '9098976756' }
    },
    publisher_city: 'balewadi',
    price: 290,
    copies: 110
  }
]
```

d. Change the city of publisher “shubham” to “Pune”.

```
exam> db.Book.updateOne({"publisher_name":"shubham"},{$set:{"publisher_city"
:"pune"}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

exam> db.Book.find()
[
  {
    _id: ObjectId("6558290e3b4c39fe061569f7"),
    book_isbn: 102,
    title: 'toc',
    publisher_name: 'sham',
    author: {
      name: 'author2',
      address: 'nashik',
      phone_no: { landline: '123-456-3453', mobile: '7080908967' }
    },
    publisher_city: 'baramati',
    price: 70,
    copies: 20
  },
  {
    _id: ObjectId("6558296e3b4c39fe061569f8"),
    book_isbn: 103,
    title: 'iot',
    publisher_name: 'shubham',
    author: {
      name: 'author3',
      address: 'dhule',
      phone_no: { landline: '345-456-3453', mobile: '9191908967' }
    },
    publisher_city: 'pune',
    price: 170,
    copies: 40
  },
  {
    _id: ObjectId("655829da3b4c39fe061569f9"),
    book_isbn: 104,
    title: 'oop',
    publisher_name: 'aniket',
    author: {
      name: 'author4',
      address: 'supe',
      phone_no: { landline: '345-546-3453', mobile: '9100008967' }
    },
    publisher_city: 'nagar',
    price: 190,
    copies: 80
  },
  {
    _id: ObjectId("65582a343b4c39fe061569fa"),
    book_isbn: 105,
    title: 'python',
    publisher_name: 'shivtej',
    author: {
      name: 'author5',
      address: 'saswad',
      phone_no: { landline: '945-546-3453', mobile: '9098976756' }
    },
    publisher_city: 'balewadi',
    price: 290,
    copies: 110
  }
]
```

e. Find the details of publisher named “shubham”.

exam> db.Book.findOne({"publisher_name":"shubham"});

```
{
  _id: ObjectId("6558296e3b4c39fe061569f8"),
  book_isbn: 103,
  title: 'iot',
  publisher_name: 'shubham',
  author: {
    name: 'author3',
    address: 'dhule',
    phone_no: { landline: '345-456-3453', mobile: '9191908967' }
  },
  publisher_city: 'pune',
  price: 170,
  copies: 40
}
```

ASSIGNMENT NO :17
Create a collection named Book. (book_isbn,title,punlisher_name,author(Name, Address, Phone No[landline, mobile]), publisher_city, price,copies)
a. Count the number of documents in the collection.
b. Arrange the documents in descending order of book_isbn.
c. Select Book Names whose title is ”DBMS” .
d. Update Book Copies as “10” whose Book Publisher is “Tata MacGraw Hill”.
Display name of publishers as per no of books published by them in ascending order.

a. Count the number of documents in the collection.

```
exam> db.Book.countDocuments()
5
```

b. Arrange the documents in descending order of book_isbn.

```
exam> db.Book.find().sort({"book_isbn":-1})
[
  {
    _id: ObjectId("65582a343b4c39fe061569fa"),
    book_isbn: 105,
    title: 'python',
    publisher_name: 'shivtej',
    author: {
      name: 'author5',
      address: 'saswad',
      phone_no: { landline: '945-546-3453', mobile: '9098976756' }
    },
    publisher_city: 'balewadi',
    price: 290,
    copies: 110
  },
  {
    _id: ObjectId("655829da3b4c39fe061569f9"),
    book_isbn: 104,
    title: 'oop',
    publisher_name: 'aniket',
    author: {
      name: 'author4',
      address: 'supe',
      phone_no: { landline: '345-546-3453', mobile: '9100008967' }
    },
    publisher_city: 'nagar',
    price: 190,
  copies: 80
},
{
  _id: ObjectId("6558296e3b4c39fe061569f8"),
  book_isbn: 103,
  title: 'iot',
  publisher_name: 'shubham',
  author: {
    name: 'author3',
    address: 'dhule',
    phone_no: { landline: '345-456-3453', mobile: '9191908967' }
  },
  publisher_city: 'pune',
  price: 170,
  copies: 40
},
{
  _id: ObjectId("6558290e3b4c39fe061569f7"),
  book_isbn: 102,
  title: 'toc',
  publisher_name: 'sham',
  author: {
    name: 'author2',
    address: 'nashik',
    phone_no: { landline: '123-456-3453', mobile: '7080908967' }
  },
  publisher_city: 'baramati',
  price: 70,
  copies: 20
},
{
  _id: ObjectId("6558306c3b4c39fe061569fb"),
  book_isbn: 101,
  title: 'dbms',
  publisher_name: 'ram',
  author: {
    name: 'author1',
    address: 'pune',
    phone_no: { landline: '123-456-7890', mobile: '1234567890' }
  },
  publisher_city: 'pune',
  price: 99,
  copies: 10
}
```

```
]
```

c. Select Book Names whose title is ”DBMS” .

```
exam> db.Book.find({"title":"dbms"})
[
  {
    _id: ObjectId("6558306c3b4c39fe061569fb"),
    book_isbn: 101,
    title: 'dbms',
    publisher_name: 'ram',
    author: {
      name: 'author1',
      address: 'pune',
      phone_no: { landline: '123-456-7890', mobile: '1234567890' }
    },
    publisher_city: 'pune',
    price: 99,
    copies: 10
  }
]
```

d. Update Book Copies as “10” whose Book Publisher is “ram”.

```
exam> db.Book.updateMany({"publisher_name":"ram"},{$set:{"copies":100}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
exam> db.Book.find({"publisher_name":"ram"})
[
  {
    _id: ObjectId("6558306c3b4c39fe061569fb"),
    book_isbn: 101,
    title: 'dbms',
    publisher_name: 'ram',
    author: {
      name: 'author1',
      address: 'pune',
      phone_no: { landline: '123-456-7890', mobile: '1234567890' }
    },
    publisher_city: 'pune',
    price: 99,
    copies: 100
  }
]
```

Display name of publishers as per no of books published by them in ascending order.

exam> db.Book.aggregate([{$group:{_id:"$publisher_name",count:{$sum:1}}},{$s
$sort:{count:1}}])

```
[
  { _id: 'sham', count: 1 },
  { _id: 'shubham', count: 1 },
  { _id: 'aniket', count: 1 },
  { _id: 'shivtej', count: 1 },
  { _id: 'ram', count: 1 }
]
```

ASSIGNMENT NO :23

Create a collection named Book. Add 5 documents in the collection with keys (book_isbn,title,punlisher_name,author(Name, Address, Phone No[landline,

mobile]), publisher_city, price,copies)
a) Select Book Names whose title is ”DBMS” .
b) Update Book Copies as “10” whose Book Publisher is “Tata MacGraw Hill”.
c) Display name of publishers as per no of books published by them in ascending order.
d) Get publisher names who published at least one book written by author name like ‘K%’.
e) Delete the book from Book table written by Author ‘author3’.

d) Get publisher names who published at least one book written by author name like ‘author3%’

```
exam> db.Book.distinct("publisher_name",{"author.name":{$regex:"author3"}})
[ 'shubham' ]
```

e) Delete the book from Book table written by Author ‘Korth’

```
exam> db.Book.deleteMany({"author.name":"author1"})
{ acknowledged: true, deletedCount: 1 }
```

ASSIGNMENT NO :18
Create a collection named “ORDERS” that contain documents of the following prototype and solve the following queries:

```
{
    cust_id: "abc123",
    ord_date: new Date("Oct 04, 2012"),
    status: 'A',
    price: 50,
    items: [
          { sku: "xxx", qty: 25, price: 1 },
          { sku: "yyy", qty: 25, price: 1 }
    ]
  },
```

a. Count all records from orders
b. Sum the price field from orders
c. For each unique cust_id, sum the price field.
d. For each unique cust_id, sum the price field, results sorted by sum.
For each unique cust_id, ord_date grouping, sum the price field

Create collection orders.

```
exam> db.createCollection("orders")
{ ok: 1 }
```

Add the documents in collection.

```
exam> db.orders.insertOne({"cust_id":"abc123","order_date":new Date("oct 04 ,2012"),
"status":"A","price":50,"items":[{sku:"xyz","qty":25,"price":1},{sku:"xxx","qty":25,"price":1}]});
{
  acknowledged: true,
  insertedId: ObjectId("655842123b4c39fe061569fc")
}
exam> db.orders.insertOne({"cust_id":"abc124","order_date":new Date("Des 04 ,2021"),
"status":"B","price":500,"items":[{sku:"dfg","qty":55,"price":2},{sku:"abc","qty":55,"price":2}]});
{
  acknowledged: true,
  insertedId: ObjectId("655842713b4c39fe061569fd")
}
exam> db.orders.insertOne({"cust_id":"abc125","order_date":new Date("Jan 04 ,2022"),
"status":"C","price":545,"items":[{sku:"jkl","qty":75,"price":3},{sku:"fgh","qty":75,"price":3}]});
{
  acknowledged: true,
  insertedId: ObjectId("655842bd3b4c39fe061569fe")
}
```

Display the data.
exam> db.orders.find()

```
[
  {
    _id: ObjectId("655842123b4c39fe061569fc"),
    cust_id: 'abc123',
    order_date: ISODate("2012-10-03T18:30:00.000Z"),
    status: 'A',
    price: 50,
    items: [
      { sku: 'xyz', qty: 25, price: 1 },
      { sku: 'xxx', qty: 25, price: 1 }
    ]
  },
  {
    _id: ObjectId("655842713b4c39fe061569fd"),
    cust_id: 'abc124',
    order_date: ISODate("1970-01-01T00:00:00.000Z"),
    status: 'B',
    price: 500,
    items: [
      { sku: 'dfg', qty: 55, price: 2 },
      { sku: 'abc', qty: 55, price: 2 }
    ]
  },
  {
    _id: ObjectId("655842bd3b4c39fe061569fe"),
    cust_id: 'abc125',
    order_date: ISODate("2022-01-03T18:30:00.000Z"),
    status: 'C',
    price: 545,
    items: [
      { sku: 'jkl', qty: 75, price: 3 },
      { sku: 'fgh', qty: 75, price: 3 }
    ]
  }
]
```

a. Count all records from orders

exam> db.orders.countDocuments()

```
3
```

b. Sum the price field from orders

```
exam> db.orders.aggregate([{$group:{_id:null,totalPrice:{$sum:"$price"}}}])
[ { _id: null, totalPrice: 1095 } ]
```

c. For each unique cust_id, sum the price field.

```
exam> db.orders.aggregate([{$group:{_id:"$cust_id",total:{$sum:"$price"}}}])

[
  { _id: 'abc123', total: 50 },
  { _id: 'abc124', total: 500 },
  { _id: 'abc125', total: 545 }
]
```

d. For each unique cust_id, sum the price field, results sorted by sum.

```
exam> db.orders.aggregate([{$group:{_id:"$cust_id", total:{$sum:"$price"}}},
{$sort:{total:1}}])
[
  { _id: 'abc123', total: 50 },
  { _id: 'abc124', total: 500 },
  { _id: 'abc125', total: 545 }
]
```

For each unique cust_id, ord_date grouping, sum the price field

```
exam>
db.orders.aggregate([{$group:{_id:{cust_id:"$cust_id",order_date:"$order_date"},total:{$
sum:"$price"}}}])
[
  {
    _id: {
      cust_id: 'abc123',
      order_date: ISODate("2012-10-03T18:30:00.000Z")
    },
    total: 50
  },
  {
    _id: {
      cust_id: 'abc124',
      order_date: ISODate("1970-01-01T00:00:00.000Z")
    },
    total: 500
  },
  {
    _id: {
      cust_id: 'abc125',
      order_date: ISODate("2022-01-03T18:30:00.000Z")
    },
    total: 545
  }
]
```

ASSIGNMENT NO :19 and 21

Create a collection named rating that contain 5 documents of the following prototype and solve the following Queries.

```
{
movie_id: 123,

user_id: 12,
title: Toy Story(1995),
status: 'A'

}
```

a) Creating an index on movie_id and sorts the keys in the index in ascending order. Verify the query plan
b) Show various indexes created on movie collection.
c) Sort movie_id in descending order.
d) Create a descending order index on movie_id to get ratings related to "Toy Story (1995)" verify the query plan.
e) Limit the number of items in the result of above query.

f) Get ratings for the movie "ICE AGE(2005)" using the descending ordered index on movie_id and explain.
g) Rebuild all indexes for the ratings collection.
h) Drop index on rating collection.
i) Create an index on movie_id and ratings fields together with movie_id (ascending order sorted) and rating (descending order sorted).
j) Create a descending order index on movie_id to get ratings related to "Toy Story (1995)" verify the query plan.

```
test> use exam
switched to db exam
```

CREATE COLLECTION RATING

```
exam> db.createCollection("rating");
{ ok: 1 }
```

INSERT 5 DOCUMENTS IN RATING COLLECTION

```
exam> db.rating.insertOne({movie_id:123,user_id:12,title:"Toy Story(1995)",s
tatus:'A'});
{
  acknowledged: true,
  insertedId: ObjectId("655896da8de0a922cbe8b966")
}
exam> db.rating.insertOne({movie_id:124,user_id:13,title:"The lion king(1994)",status:'A'});
{
  acknowledged: true,
  insertedId: ObjectId("655897028de0a922cbe8b967")
}
exam> db.rating.insertOne({movie_id:125,user_id:14,title:"finding nemo(2003)",status:'A'});
{
  acknowledged: true,
  insertedId: ObjectId("6558972b8de0a922cbe8b968")
}
exam> db.rating.insertOne({movie_id:126,user_id:15,title:"shrek (2001)",status:'A'});
{
  acknowledged: true,
  insertedId: ObjectId("655897758de0a922cbe8b969")
}
exam> db.rating.insertOne({movie_id:127,user_id:16,title:"frozen (2013)",status:'A'});
{
  acknowledged: true,
  insertedId: ObjectId("655897988de0a922cbe8b96a")
}
```

exam> db.rating.find()

```
[
  {
    _id: ObjectId("655896da8de0a922cbe8b966"),
    movie_id: 123,
    user_id: 12,
    title: 'Toy Story(1995)',
    status: 'A'
  },
  {
    _id: ObjectId("655897028de0a922cbe8b967"),
    movie_id: 124,
    user_id: 13,
    title: 'The lion king(1994)',
    status: 'A'
  },
  {
    _id: ObjectId("6558972b8de0a922cbe8b968"),
    movie_id: 125,
    user_id: 14,
    title: 'finding nemo(2003)',
    status: 'A'
  },
  {
    _id: ObjectId("655897758de0a922cbe8b969"),
    movie_id: 126,
    user_id: 15,
    title: 'shrek (2001)',
    status: 'A'
  },
  {
    _id: ObjectId("655897988de0a922cbe8b96a"),
    movie_id: 127,
    user_id: 16,
    title: 'frozen (2013)',
    status: 'A'
  }
]
```

a) Creating an index on movie_id and sorts the keys in the index in ascending order. Verify the query plan

```
exam> db.rating.createIndex({movie_id:1});
Movie_id_1
```

Verify the query plan

```
exam> db.rating.find({movie_id:123}).explain("executionStatus");
```

b) Show various indexes created on movie collection.

```
exam> db.rating.getIndexes();
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { movie_id: 1 }, name: 'movie_id_1' }
]
```

c) Sort movie_id in descending order.

```
exam> db.rating.find().sort({movie_id:-1});
[
  {
    _id: ObjectId("655897988de0a922cbe8b96a"),
    movie_id: 127,
    user_id: 16,
    title: 'frozen (2013)',
    status: 'A'
  },
  {
    _id: ObjectId("655897758de0a922cbe8b969"),
    movie_id: 126,
    user_id: 15,
    title: 'shrek (2001)',
    status: 'A'
  },
  {
    _id: ObjectId("6558972b8de0a922cbe8b968"),
    movie_id: 125,
    user_id: 14,
    title: 'finding nemo(2003)',
    status: 'A'
  },
  {
    _id: ObjectId("655897028de0a922cbe8b967"),
    movie_id: 124,
    user_id: 13,
    title: 'The lion king(1994)',
    status: 'A'
  },
  {
    _id: ObjectId("655896da8de0a922cbe8b966"),
    movie_id: 123,
    user_id: 12,
    title: 'Toy Story(1995)',
    status: 'A'
  }
]
```

d) Create a descending order index on movie_id to get ratings related to "Toy Story (1995)" verify the query plan.

```
exam> db.rating.createIndex({ movie_id: -1 }, { partialFilterExpression: { title: "Toy Story (1995)"
} });
Movie_id_-1
```

```
exam>  db.rating.getIndexes();
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { movie_id: 1 }, name: 'movie_id_1' },
  {
    v: 2,
    key: { movie_id: -1 },
    name: 'movie_id_-1',
    partialFilterExpression: { title: 'Toy Story (1995)' }
  }
]
```

verify the query plan.

```
db.rating.find({ title: "Toy Story (1995)" }).sort({ movie_id: -1 }).explain("executionStats");
```

e) Limit the number of items in the result of above query.

```
exam> db.rating.find({title:'Toy story(1995)'},{rating:1}).sort({movie_id:-1}).limit(2);
```

f) Get ratings for the movie “finding nemo(2003)” using the descending ordered index on movie_id and explain.

```
db.rating.find({ "title": "finding nemo(2003)" }).sort({ "movie_id": -1 }).explain("executionStats");
```

g) Rebuild all indexes for the ratings collection.

```
exam> db.rating.reIndex();
{
  nIndexesWas: 3,
  nIndexes: 3,
  indexes: [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    { v: 2, key: { movie_id: 1 }, name: 'movie_id_1' },
    {
      v: 2,
      key: { movie_id: -1 },
      name: 'movie_id_-1',
      partialFilterExpression: { title: 'Toy Story (1995)' }
    }
  ],
  ok: 1
}
```

h) Drop index on rating collection.

```
exam>  db.rating.dropIndexes();
{
  nIndexesWas: 3,
  msg: 'non-_id indexes dropped for collection',
  ok: 1
}
```

i) Create an index on movie_id and ratings fields together with movie_id (ascending order sorted) and rating (descending order sorted).

```
exam>  db.rating.ensureIndex({movie_id:1,rating:-1});
[ 'movie_id_1_rating_-1' ]
exam>  db.rating.getIndexes();
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  {
    v: 2,
    key: { movie_id: 1, rating: -1 },
    name: 'movie_id_1_rating_-1'
  }
```

j) Create a descending order index on movie_id to get ratings related to “Toy Story (1995)” verify the query plan.

```
exam> db.rating.createIndex({ "movie_id": -1 });
Movie_id_-1

db.rating.find({ "title": "Toy Story (1995)" }).sort({ "movie_id": -1 }).explain("executionStats");
```

ASSIGNMENT NO :20

Design a map-reduce operations on a collection “orders” that contains documents of the following prototype. Solve the following .

```
{
cust_id: ‘abc123’
ord_date: new Date(Oct 04, 2012),
status: ‘A’
price: 25,
gender :’F’,
rating: 1
}
```

a) Count the number of female (F) and male (M) respondents in the orders collection
b) Count the number of each type of rating (1, 2, 3, 4 or 5) for each orders

CREATE COLLECTION ORDERS

```
exam> db.createCollection("orders");
{ ok: 1 }
```

INSERT 5 DOCUMENTS

```
exam> db.orders.insertOne({cust_id:"abc123",order_date:new Date("Oct 04,2012"),status:'A',
price:50, gender:'F', rating:1})
{
  acknowledged: true,
  insertedId: ObjectId("6558aa868de0a922cbe8b96b")
}
exam> db.orders.insertOne({cust_id:"def456",order_date:new Date("Sep 15,2012"),status:'A',
price:30, gender:'M', rating:3})
{
  acknowledged: true,
  insertedId: ObjectId("6558aac48de0a922cbe8b96c")
}
exam> db.orders.insertOne({cust_id:"ghi789",order_date:new Date("Nov 20,2012"),status:'A',
price:15, gender:'F', rating:5})
{
  acknowledged: true,
  insertedId: ObjectId("6558ab0d8de0a922cbe8b96d")
}
exam> db.orders.insertOne({cust_id:"jkl012",order_date:new Date("Oct 30,2012"),status:'B',
price:40, gender:'M', rating:5})
{
  acknowledged: true,
  insertedId: ObjectId("6558ab688de0a922cbe8b96e")
}
exam> db.orders.insertOne({cust_id:"mno234",order_date:new Date("Dec 30,2012"),status:'B',
price:20, gender:'M', rating:4})
{
  acknowledged: true,
  insertedId: ObjectId("6558ab978de0a922cbe8b96f")
}
```

DISPLAY THE DATA

```
exam> db.orders.find()
[
 {
   _id: ObjectId("6558aa868de0a922cbe8b96b"),
   cust_id: 'abc123',
   order_date: ISODate("2012-10-03T18:30:00.000Z"),
   status: 'A',
   price: 50,
   gender: 'F',
   rating: 1
 },
 {
   _id: ObjectId("6558aac48de0a922cbe8b96c"),
   cust_id: 'def456',
   order_date: ISODate("2012-09-14T18:30:00.000Z"),
   status: 'A',
   price: 30,
   gender: 'M',
   rating: 3
 },
 {
   _id: ObjectId("6558ab0d8de0a922cbe8b96d"),
   cust_id: 'ghi789',
   order_date: ISODate("2012-11-19T18:30:00.000Z"),
   status: 'A',
   price: 15,
   gender: 'F',
   rating: 5
 },
 {
   _id: ObjectId("6558ab688de0a922cbe8b96e"),
   cust_id: 'jkl012',
   order_date: ISODate("2012-10-29T18:30:00.000Z"),
   status: 'B',
   price: 40,
   gender: 'M',
   rating: 5
 },
 {
   _id: ObjectId("6558ab978de0a922cbe8b96f"),
   cust_id: 'mno234',
   order_date: ISODate("2012-12-29T18:30:00.000Z"),
   status: 'B',
   price: 20,
   gender: 'M',
   rating: 4
 }
]
```

a) Count the number of female (F) and male (M) respondents in the orders collection

ASSIGNMENT NO :24

Consider following structure for MongoDB collections and write a query for following requirements in MongoDB
Teachers(Tname, dno, experience, salary, date_of joining)
Students(Sname, roll_no, class)
i) Write a MongoDB query to create above collections &amp; for insertion of some sample documents.
ii) Find the information about all teachers of dno = 2 and having salary greater than or equal to 10,000/-
iii) Find the student information having roll_no = 2 or Sname = Anil
iv) Display Total no of Students of TE Class
V) update salary as 5% increment of teacher whose experience is &gt;10 years.

i) Write a MongoDB query to create above collections &amp; for insertion of some sample documents.

```
exam> db.createCollection("teachers")
{ ok: 1 }

exam> db.teachers.insertMany([
... {Tname: "John", dno: 1, experience: 12, salary: 12000, date_of_joining: new
Date("2010-05-20")},{ Tname: "Alice", dno: 2, experience: 8, salary: 11000, date_of_joining: new
Date("2015-02-10") },{ Tname: "Bob", dno: 2, experiexperience: 15, salary: 15000,
date_of_joining: new Date("2005-09-15") }]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6558de4bcedd1deec6e4d109"),
    '1': ObjectId("6558de4bcedd1deec6e4d10a"),
    '2': ObjectId("6558de4bcedd1deec6e4d10b")
  }
}
```

exam> db.teachers.find()

```
[
  {
    _id: ObjectId("6558de4bcedd1deec6e4d109"),
    Tname: 'John',
    dno: 1,
    experience: 12,
    salary: 12000,
    date_of_joining: ISODate("2010-05-20T00:00:00.000Z")
  },
  {
    _id: ObjectId("6558de4bcedd1deec6e4d10a"),
    Tname: 'Alice',
    dno: 2,
    experience: 8,
    salary: 11000,
    date_of_joining: ISODate("2015-02-10T00:00:00.000Z")
  },
  {
    _id: ObjectId("6558de4bcedd1deec6e4d10b"),
    Tname: 'Bob',
    dno: 2,
    experience: 15,
    salary: 15000,
    date_of_joining: ISODate("2005-09-15T00:00:00.000Z")
  }
]
```

exam> db.createCollection("Students");

```
{ ok: 1 }
```

```
exam> db.Students.insertMany([{ Sname: "Anil", roll_no: 1, class: "TE" },{ Sname: "Sarah",
roll_no: 2, class: "TE" }, { Sname: "David", roll_no: 3, clasclass: "SE" }]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6558deb5cedd1deec6e4d10c"),
    '1': ObjectId("6558deb5cedd1deec6e4d10d"),
    '2': ObjectId("6558deb5cedd1deec6e4d10e")
  }
}

exam> db.Students.find()
[
  {
    _id: ObjectId("6558deb5cedd1deec6e4d10c"),
    Sname: 'Anil',
    roll_no: 1,
    class: 'TE'
  },
  {
    _id: ObjectId("6558deb5cedd1deec6e4d10d"),
    Sname: 'Sarah',
    roll_no: 2,
    class: 'TE'
  },
  {
    _id: ObjectId("6558deb5cedd1deec6e4d10e"),
    Sname: 'David',
    roll_no: 3,
    class: 'SE'
  }
]
```

ii) Find the information about all teachers of dno = 2 and having salary greater than or equal to 10,000/-

```
exam> db.teachers.find({dno:2,salary:{$gte:10000}});
[
  {
    _id: ObjectId("6558de4bcedd1deec6e4d10a"),
    Tname: 'Alice',
    dno: 2,
    experience: 8,
    salary: 11000,
    date_of_joining: ISODate("2015-02-10T00:00:00.000Z")
  },
  {
    _id: ObjectId("6558de4bcedd1deec6e4d10b"),
    Tname: 'Bob',
    dno: 2,
    experience: 15,
    salary: 15000,
    date_of_joining: ISODate("2005-09-15T00:00:00.000Z")
  }
]
```

iii) Find the student information having roll_no = 2 or Sname = Anil

```
exam> db.Students.find({ $or: [{ roll_no: 2 }, { Sname: "Anil" }] });
[
  {
    _id: ObjectId("6558deb5cedd1deec6e4d10c"),
    Sname: 'Anil',
    roll_no: 1,
    class: 'TE'
  },
  {
    _id: ObjectId("6558deb5cedd1deec6e4d10d"),
    Sname: 'Sarah',
    roll_no: 2,
    class: 'TE'
  }
]
```

iv) Display Total no of Students of TE Class

```
exam> db.Students.count({class:"TE"});
2
```

V) update salary as 5% increment of teacher whose experience is &gt;10 years.

```
exam> db.teachers.updateMany({experience:{$gt:10}},{$mul:{salary:1.05}});
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 2,
  modifiedCount: 2,
  upsertedCount: 0
}
exam> db.teachers.find()
[
  {
    _id: ObjectId("6558de4bcedd1deec6e4d109"),
    Tname: 'John',
    dno: 1,
    experience: 12,
    salary: 12600,
    date_of_joining: ISODate("2010-05-20T00:00:00.000Z")
  },
  {
    _id: ObjectId("6558de4bcedd1deec6e4d10a"),
    Tname: 'Alice',
    dno: 2,
    experience: 8,
    salary: 11000,
    date_of_joining: ISODate("2015-02-10T00:00:00.000Z")
  },
  {
    _id: ObjectId("6558de4bcedd1deec6e4d10b"),
    Tname: 'Bob',
    dno: 2,
    experience: 15,
    salary: 15750,
    date_of_joining: ISODate("2005-09-15T00:00:00.000Z")
  }
]
```

ASSIGNMENT NO :26

Create the following table with the fields given below : PRODUCT (P_ID, Model, Price, Name, Date_of Manufacture, Date_of Expiry)
(a) Display name and date_of expiry of all the products whose price is more than 500.
(b) Display name, product_ID and price of all the products whose date_of manufacture is after '01-01-2018'.
(c) Display name and date_of manufacture and date- of expiry of all the products whose price is between 5,000 and 10,000.
(d) Display name, product_ID and model of all the products which are going to expire after two months from today.

```
mysql> use dbms;
Database changed
mysql> create table product(p_id int primary key, model varchar(50),price decimal(10,2),name
varchar(100),date_of_manufacture date,date_of_expiry date);Query OK, 0 rows affected (0.14
sec)


mysql> describe table product;
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table   | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered |
Extra |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | product | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |
100.00 | NULL  |
+----+-------------+---------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.02 sec)

mysql> insert into product(p_id,model,price,name,date_of_manufacture,date_of_expiry)values
    -> (1, 'Model1', 750.00, 'Product 1', '2022-01-05', '2023-01-05'),
    -> (2, 'Model2', 300.00, 'Product 2', '2022-02-10', '2023-02-10'),
    -> (3, 'Model3', 5500.00, 'Product 3', '2022-03-15', '2023-03-15'),
    -> (4, 'Model4', 8000.00, 'Product 4', '2022-04-20', '2023-04-20'),
    -> (5, 'Model5', 9500.00, 'Product 5', '2022-05-25', '2023-05-25');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> select * from product;
+------+--------+---------+-----------+---------------------+----------------+
| p_id | model  | price   | name      | date_of_manufacture | date_of_expiry |
+------+--------+---------+-----------+---------------------+----------------+
|    1 | Model1 |  750.00 | Product 1 | 2022-01-05          | 2023-01-05     |
|    2 | Model2 |  300.00 | Product 2 | 2022-02-10          | 2023-02-10     |
|    3 | Model3 | 5500.00 | Product 3 | 2022-03-15          | 2023-03-15     |
|    4 | Model4 | 8000.00 | Product 4 | 2022-04-20          | 2023-04-20     |
|    5 | Model5 | 9500.00 | Product 5 | 2022-05-25          | 2023-05-25     |
+------+--------+---------+-----------+---------------------+----------------+
5 rows in set (0.01 sec)
```

(a) Display name and date_of expiry of all the products whose price is more than 500.

```
mysql> select name,date_of_expiry from product where price>500;
+-----------+----------------+
| name      | date_of_expiry |
+-----------+----------------+
| Product 1 | 2023-01-05     |
| Product 3 | 2023-03-15     |
| Product 4 | 2023-04-20     |
| Product 5 | 2023-05-25     |
+-----------+----------------+
4 rows in set (0.02 sec)
```

(b) Display name, product_ID and price of all the products whose date_of manufacture is after
'01-01-2018'.

```
mysql> select name, p_id, model
    -> from product
    -> where date_of_manufacture > '2022-03-14';
+-----------+------+--------+
| name      | p_id | model  |
+-----------+------+--------+
| Product 3 |    3 | Model3 |
| Product 4 |    4 | Model4 |
| Product 5 |    5 | Model5 |
+-----------+------+--------+
3 rows in set (0.00 sec)
```

(c) Display name and date_of manufacture and date- of expiry of all the products whose price is
between 5,000 and 10,000.

```
mysql> select name, p_id, model
    -> from product
    -> where price between 5000 AND 10000;
+-----------+------+--------+
| name      | p_id | model  |
+-----------+------+--------+
| Product 3 |    3 | Model3 |
| Product 4 |    4 | Model4 |
| Product 5 |    5 | Model5 |
+-----------+------+--------+
3 rows in set (0.01 sec)
```

(d) Display name, product_ID and model of all the products which are going to expire after two
months from today.

```
mysql> select name, p_id, model
    -> from product
    -> where date_of_expiry >date_add(curdate(),interval 2 month);
Empty set (0.01 sec)
```

PRACTICAL NO 31

```
exam> db.createCollection("restaurants");
{ ok: 1 }
exam> db.restaurants.insertMany([
... {
... "address": {
... "building": "1007",
... "coord": [-73.856077, 40.848447],
... "street": "Morris Park Ave",
... "zipcode": "10462"
... },
... "borough": "Bronx",
...  "cuisine": "Bakery",
...  "grades": [
... { "date": new Date("2014-03-03"), "grade": "A", "score": 2 },
... { "date": new Date("2013-09-11"), "grade": "A", "score": 6 },
... { "date": new Date("2013-01-24"), "grade": "A", "score": 10 },
... { "date": new Date("2011-11-23"), "grade": "A", "score": 9 },
... { "date": new Date("2011-03-10"), "grade": "B", "score": 14 }
...  ],
...  "name": "Morris Park Bake Shop",
... "restaurant_id": "30075445"
... },]);
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("655996084bea37949925c1f4") }
}
```

a.Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the documents in the collection restaurant.

```
exam> db.restaurants.find({}, { "restaurant_id": 1, "name": 1, "borough": 1,
 "cuisine": 1, "_id": 0 });
[
  {
    borough: 'Bronx',
    cuisine: 'Bakery',
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
]
```

b. Write a MongoDB query to display the fields restaurant_id, name, borough and cuisine for all the documents in the collection restaurant.

```
exam> db.restaurants.find({}, { "restaurant_id": 1, "name": 1, "borough": 1,
 "cuisine": 1, "_id": 0 }).pretty();
[
  {
    borough: 'Bronx',
    cuisine: 'Bakery',
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
]
```

c. Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant.

```
exam> db.restaurants.find({}, { "restaurant_id": 1, "name": 1, "borough": 1,
 "address.zipcode": 1, "_id": 0 });
[
  {
    address: { zipcode: '10462' },
    borough: 'Bronx',
    name: 'Morris Park Bake Shop',
    restaurant_id: '30075445'
  }
]
```