```
//a: This Will Create StudentMaster and insert  10 records In StudentMaster Collection
db.StudentMaster.insertMany([
  {
    _id: 1,
    StudentRollNo: 101,
    StudentName: "Smit Joshi",
    Grade: "XII",
    Hobbies: ["Coding", "Travelling"],
    DOJ: new Date("2012-07-28"),
  },
  {
    _id: 2,
    StudentRollNo: 102,
    StudentName: "Ajay Rathod",
    Grade: "VI",
    Hobbies: ["Coding", "Chess"],
    DOJ: new Date("2013-08-28"),
  },
  {
    _id: 3,
    StudentRollNo: 101,
    StudentName: "Mayur Joshi",
    Grade: "VII",
    Hobbies: ["Dancing", "Chess"],
    DOJ: new Date("2014-09-28"),
  },
  {
    _id: 4,
    StudentRollNo: 101,
    StudentName: "Isha Sharma",
    Grade: "XI",
    Hobbies: ["Dancing", "Chess"],
    DOJ: new Date("2015-10-28"),
  },
  {
    _id: 5,
    StudentRollNo: 101,
    StudentName: "Bhavesh",
    Grade: "VI",
    Hobbies: ["Coding", "Chess"],
    DOJ: new Date("2016-07-28"),
  },
  {
    _id: 6,
    StudentRollNo: 101,
    StudentName: "Hiten Joshi",
    Grade: "IV",
    Hobbies: ["Dancing", "Chess"],
    DOJ: new Date("2017-07-28"),
  },
  {
    _id: 7,
    StudentRollNo: 101,
    StudentName: "Vijay Joshi",
```

```
    Grade: "IV",
    Hobbies: ["Dancing", "Chess"],
    DOJ: new Date("2017-07-28"),
  },
  {
    _id: 8,
    StudentRollNo: 101,
    StudentName: "Palak",
    Grade: "IV",
    Hobbies: ["Dancing", "Modeling"],
    DOJ: new Date("2018-07-28"),
  },
  {
    _id: 9,
    StudentRollNo: 101,
    StudentName: "Vishva Joshi",
    Grade: "IV",
    Hobbies: ["Teaching", "Chess"],
    DOJ: new Date("2019-07-28"),
  },
  {
    _id: 10,
    StudentRollNo: 101,
    StudentName: "Poojan",
    Grade: "VII",
    Hobbies: ["Dancing", "Chess"],
    DOJ: new Date("2020-07-28"),
  },
]);

// b: Find Doc where StudentName has Value Ajay Rathod"
db.StudentMaster.find({
  StudentName: "Ajay Rathod",
});

// c: find all docs withoud _id field
db.StudentMaster.find({}, { _id: 0 });

// d: retrive only StudentName and Grade
db.StudentMaster.find({}, { StudentName: 1, Grade: 1, _id: 0 });

// e:  retrive only StudentName and Grade who is having _id:1
db.StudentMaster.find({ _id: 1 }, { StudentName: 1, Grade: 1 });

// f: Add New Field Address to collection
db.StudentMaster.updateMany({}, { $set: { Address: "" } });

// g: find docs where Grade is VII
db.StudentMaster.find({ Grade: "VII" });

// h: find docs where Grade is not VII
db.StudentMaster.find({
  $nor: [{ Grade: "VII" }],
});
```

```
// i: find docs where Hobbies is Set to Either "Chess" or "Dancing"
db.StudentMaster.find({
  $or: [{ Hobbies: { $in: ["Chess", "Dancing"] } }],
});

// j: find docs where Hobbies is Set to Neither "Chess" or "Dancing"
db.StudentMaster.find({
  $nor: [{ Hobbies: { $in: ["Chess", "Dancing"] } }],
});

// k: find docs where StudentName Begins With 'M'
db.StudentMaster.find({
  StudentName: /^M/,
});

// l: find docs Where StudentName has 'e' in any Position
db.StudentMaster.find({
  StudentName: { $regex: "e" },
});

// m: find docs where StudentName ends With 'a'
db.StudentMaster.find({
  StudentName: /a$/,
});

// n: find total number of docs present in Collection
db.StudentMaster.find().count();

// o: find total number of docs With Grade 'VII' present in Collection
db.StudentMaster.find({ Grade: "VII" }).count();

// p: sort the docs in Ascending order based on StudentName
db.StudentMaster.find().sort({ StudentName: 1 });

// q: Display the last Two records
db.StudentMaster.find().sort({ _id: -1 }).limit(2);

/* Other Queries */

// find Record By Date
db.StudentMaster.find({
  DOJ: ISODate("2020-07-28"),
});

// find Record by Given Range Of Dates
db.StudentMaster.find({
  DOJ: {
    $lte: ISODate("2020-07-28"),
    $gte: ISODate("2016-07-28"),
  },
});
```

```
// Movies Database

db.Movies.insertMany([
  {
    Titles: "The Bradman",
    Directors: "Shinchgan",
    Years: 2010,
    Actors: ["Chnadler", "Himan"],
  },
  {
    Titles: "The Hobbit:An Unexpected Journey",
    Directors: "Quentin Tarantino",
    Years: 2012,
    Actors: ["Brad Pitt", "Himan"],
  },
  {
    Titles: "The Hobbit:The Desolation of Smaug",
    Directors: "Quentin Tarantino",
    Years: 1999,
    Actors: ["Brad Shitt", "Jhon Wick"],
  },
  {
    Titles: "Dragonss",
    Directors: "Rohit Shetty",
    Years: 2015,
    Actors: ["Brad Pitt", "Himan", "Mariano"],
  },
  {
    Titles: "Cars",
    Directors: "Will Smith",
    Years: 1990,
    Actors: ["Tonyy Lee", "Rick Flair", "Ronda-rousy"],
  },
  {
    Titles: "Pulp Fiction",
    Directors: "Quentin Tarantino",
    Years: 2010,
    Actors: ["Jhonny Liver", "Himan"],
  },
  {
    Titles: "Jumanji",
    Directors: "Anurag Kashyup",
    Years: 2011,
    Actors: ["Brad Pitt", "Aarnold"],
  },
  {
    Titles: "Zathura:Space-Adventure",
    Directors: "Quentin Tartos",
    Years: 1995,
    Actors: ["Hulk", "Hogan"],
  },
  {
    Titles: "Wanted",
    Directors: "Tarantino",
```

```
    Years: 2005,
    Actors: ["Seth", "Rolins"],
  },
  {
    Titles: "Kedarnath",
    Directors: "Abhijet Dalal",
    Years: 2010,
    Actors: ["Sushant", "Sara"],
  },
]);

// 1. Retrive all Documents
db.Movies.find();

// 2. Retive All Documents with Director set to Quentin Tarantino
db.Movies.find({Directors:"Quentin Tarantino"});

// 3. Retrive all documents where actors include "Bread pitt"
db.Movies.find({Actors:{$in:['Brad Pitt']}});

// 4. retrive all movies released before year 2000 or after 2010
db.Movies.find({$or:[{Years:{$lt:2000}},{Years:{$gt:2010}}]});

// 5. add synopsis to The Hobbit:An Unexpected Journey
db.Movies.updateOne({Titles:"The Hobbit:An Unexpected Journey"},{$set:{synopsis:'"The
Hobbit:An Unexpected Journey":"A reluctant hobbit,Bilbo Baggins, sets out to the Lonely
Mountain with a spirited group of dwarves to reclaim their mountain home - and the gold
within it - from the dragon Smaug."'}});

// 6. add synopsis to The Hobbit:The Desolation of Smaug
db.Movies.updateOne({Titles:"The Hobbit:The Desolation of Smaug"},{$set:{synopsis:"The
dwarves, along with Bilbo Baggins and Gandalf Grey,continue their quest to reclaim Erebor,
their homeland, from Smaug. Bilbo Baggins is in possession of a mysterious and magical
ring."}});

// 7. add an actor named Samuel L. Jackson to movie Pulp Fiction
db.Movies.updateOne({Titles:"Pulp Fiction"},{$push:{Actors:{$each:['Samuel L.
Jackson']}}});

// 8. find all movies that have synopsis that contains word "Biblo"
db.Movies.find({synopsis:{$regex:"Bilbo"}});

// 9. find all movies that have synopsis that contains word "Gandlf"
db.Movies.find({synopsis:{$regex:"Gandalf"}});

// 10. Find all movies that have a synopsis that contains the word "Bilbo" and not the
word "Gandalf"
db.Movies.find({$and: [
  {synopsis: {$regex: "Bilbo"}},
  { $nor: [ {synopsis: {$regex: "Gandalf"}}]}
]});

// 11. Find all movies that have synopsis that contains word "dwarves" or "hobbit"
db.Movies.find({$or:[
```

```
  {synopsis:{$regex:"dwarves"}},
  {synopsis:{$regex:"hobbit"}}
]});

// 12. find all movies that have synopsis that contains word "gold" and "dragon"
db.Movies.find({$and:[
  {synopsis:{$regex:"gold"}},
  {synopsis:{$regex:"dragon"}}
]});

// 13. delete the movie Pee Wee Herman's Big Adventure
db.Movies.deleteOne({Titles:"Pee Wee Herman's Big Adventure"});


// Product Collection
db.Products.insertMany([
  {
    "product_id": 1,
    "product_name": "Pencil",
    "product_type": "Stationary",
    "cost_unit": 9,
    "qty_in_stock": 100
  },
  {
    "product_id": 2,
    "product_name": "Notebook",
    "product_type": "Stationary",
    "cost_unit": 59,
    "qty_in_stock": 50
  },
  {
    "product_id": 3,
    "product_name": "Laptop",
    "product_type": "Electronics",
    "cost_unit": 30999,
    "qty_in_stock": 20
  },
  {
    "product_id": 4,
    "product_name": "Tablet",
    "product_type": "Electronics",
    "cost_unit": 38999,
    "qty_in_stock": 50
  },
  {
    "product_id": 5,
    "product_name": "Chair",
    "product_type": "Furniture",
    "cost_unit": 2999,
    "qty_in_stock": 100
  },
  {
    "product_id": 6,
    "product_name": "Book",
```

```
    "product_type": "Stationary",
    "cost_unit": 299,
    "qty_in_stock": 100
  },
  {
    "product_id": 7,
    "product_name": "Mobile Phone",
    "product_type": "Electronics",
    "cost_unit": 29599,
    "qty_in_stock": 50
  },
  {
    "product_id": 8,
    "product_name": "Sofa",
    "product_type": "Home Decor",
    "cost_unit": 7999,
    "qty_in_stock": 100
  },
  {
    "product_id": 9,
    "product_name": "Bed",
    "product_type": "Furniture",
    "cost_unit": 8999,
    "qty_in_stock": 100
  },
  {
    "product_id": 10,
    "product_name": "Headphones",
    "product_type": "Electronics",
    "cost_unit": 1599,
    "qty_in_stock": 100
  }
]);

// 1. display unique list of product categories
db.Products.aggregate({
  $group:{
    _id:"",
    product_categories:{$addToSet:"$product_type"}
  }
});

// 2. display the second document on the collection with only product_name,product_type
and cost_unit fields
db.Products.find({},{_id:0,product_name:1,product_type:1,cost_unit:1}).limit(1).skip(1);

// 3. display costliest product for all categories
db.Products.aggregate({ $group: { _id: "$product_type", costliest: { $max: "$cost_unit" },
product_name:{$first:"$product_name"} } });

// 4. add new field discount_offered for those furniture having price between 15k to 20k
db.Products.updateMany({ $and: [
  {"product_type": "Furniture"},
  {"cost_unit": { $lt: 15000 } },
```

```
  {"cost_unit": { $gt: 20000 } }
  ]},{
  $set: {
      "discount_offered": 0
  }
});


// 5. display those electronics products price having lover then 25k , store in array and
print using appropriate method
var arr = db.Products.find({ "cost_unit": { $lt: 25000 } });
var electronics=[];
arr.forEach(val => {
  electronics.push(val)
});
// arr.toArray();
// print(arr.toArray());
print(electronics)

// 6. update cost_unit of Electronics category by giving discount of 1000
db.Products.updateMany({"product_type":"Electronics"},{$inc:{cost_unit:-1000}});


// 7. delete qty_in_stock for all whose cost lies b/w 500 t0 800
db.Products.updateMany({$and:[
  {"cost_unit":{$gte:500}},
  {"cost_unit":{$lte:800}}
]},{$unset:"qty_in_stock"});


// 8. add new column "Reorder Qty" to the collection
db.Products.updateMany({},{$set:{"Reorder_Qty":""}});

// 9. Using map reduce display the product details such as Product name and cost for
products belonging to Stationary Category
  //Mapper
var mapFunction = function() {
  if (this.product_type === "Stationary") {
      emit(this.product_name, this.cost_unit);
  }
}
//reducer
var reduceFunction = function(key, value) {
  return Array(value)
}
//mapReduce
db.Products.mapReduce(mapFunction, reduceFunction, {
  out: "products"
});
//output variable
db.products.find();

// 10. Create an index on given collection
db.Products.createIndex({product_name:1});
```

```
/*

Other Queries

*/

// to create index
db.collection.createIndex({ field: 1 });
db.Books.createIndex({ Book_Code: 1 });

// to delete index
db.Books.dropIndex({ Book_Code: 1 });

// executionstates
db.collection.find().explain("executionStats");

// curser => is a pointer to the resultset
var variable = db.Product.find({ qty_in_stock: { $gt: 25 } });
// will iterate and print the values found in curser
while (variable.hasNext()) {
  print(variable.next());
}

//courser methods
curser.count();
curser.explain();
curser.forEach();

// foeEach exam
db.product_purchase.find().forEach(function (doc) {
  print("product: " + doc.product_name);
});
// Enterprise NoSQL_Collage> variable.forEach(function(doc){print("product:
"+doc.product_name);})
// product: Notebook
// product: Chair
// product: Book
// product: Sofa

curser.hasNext();
curser.limit();  // limit will only work with db.collection.find() method not after that
// still if you tend to use "MongoCursorInUseError: Cursor is already initialized" error
will be shown

curser.next();
curser.pretty();
curser.skip(); // same error as limit as it can only be used with find() method
// "MongoCursorInUseError: Cursor is already initialized"

curser.sort();  // same error only be used with find()
```

```
curser.toArray();
curser.init[0]; // Error
curser.open; //will open the curser
curser.close; // will close the curser

// using toArray()
var variable = db.Product.find({ qty_in_stock: { $gt: 25 } });
var temp = variable.toArray();
for (i = 0; i < temp.length; i++) {
  print(temp[i]);
}

// diff b/v cursur & find()

db.collection.find(); // will return 20 documents at first after that you've to write it
// where as cursor returns the whole documents
// cursor is useful when you have to get the data which has documents in thousends or
// may be larger where as find() which also returns the curser gives the 20 documents
// which is usefull when you don't need the whole docs on the first run
```