```
db.test.insertMany([
  {
    "_id" : 1,
    "name" : {
      "first" : "John",
      "last" : "Backus"
    },
    "birth" : ISODate("1924-12-03T05:00:00Z"),
    "death" : ISODate("2007-03-17T04:00:00Z"),
    "contribs" : [
      "Fortran",
      "ALGOL",
      "Backus-Naur Form",
      "FP"
    ],
    "awards" : [
      {
        "award" : "W.W. McDowell Award",
        "year" : 1967,
        "by" : "IEEE Computer Society"
      },
      {
        "award" : "National Medal of Science",
        "year" : 1975,
        "by" : "National Science Foundation"
      },
      {
        "award" : "Turing Award",
        "year" : 1977,
        "by" : "ACM"
      },
      {
        "award" : "Draper Prize",
        "year" : 1993,
        "by" : "National Academy of Engineering"
      }
    ]
  },
  {
    "_id" : ObjectId("51df07b094c6acd67e492f41"),
    "name" : {
      "first" : "John",
      "last" : "McCarthy"
    },
    "birth" : ISODate("1927-09-04T04:00:00Z"),
    "death" : ISODate("2011-12-24T05:00:00Z"),
    "contribs" : [
      "Lisp",
      "Artificial Intelligence",
      "ALGOL"
    ],
    "awards" : [
```

```
      {
        "award" : "Turing Award",
        "year" : 1971,
        "by" : "ACM"
      },
      {
        "award" : "Kyoto Prize",
        "year" : 1988,
        "by" : "Inamori Foundation"
      },
      {
        "award" : "National Medal of Science",
        "year" : 1990,
        "by" : "National Science Foundation"
      }
    ]
},
{
    "_id" : 3,
    "name" : {
      "first" : "Grace",
      "last" : "Hopper"
    },
    "title" : "Rear Admiral",
    "birth" : ISODate("1906-12-09T05:00:00Z"),
    "death" : ISODate("1992-01-01T05:00:00Z"),
    "contribs" : [
      "UNIVAC",
      "compiler",
      "FLOW-MATIC",
      "COBOL"
    ],
    "awards" : [
      {
        "award" : "Computer Sciences Man of the Year",
        "year" : 1969,
        "by" : "Data Processing Management Association"
      },
      {
        "award" : "Distinguished Fellow",
        "year" : 1973,
        "by" : " British Computer Society"
      },
      {
        "award" : "W. W. McDowell Award",
        "year" : 1976,
        "by" : "IEEE Computer Society"
      },
      {
        "award" : "National Medal of Technology",
        "year" : 1991,
        "by" : "United States"
```

```
      }
    ]
},
{
    "_id" : 4,
    "name" : {
        "first" : "Kristen",
        "last" : "Nygaard"
    },
    "birth" : ISODate("1926-08-27T04:00:00Z"),
    "death" : ISODate("2002-08-10T04:00:00Z"),
    "contribs" : [
        "OOP",
        "Simula"
    ],
    "awards" : [
        {
            "award" : "Rosing Prize",
            "year" : 1999,
            "by" : "Norwegian Data Association"
        },
        {
            "award" : "Turing Award",
            "year" : 2001,
            "by" : "ACM"
        },
        {
            "award" : "IEEE John von Neumann Medal",
            "year" : 2001,
            "by" : "IEEE"
        }
    ]
},
{
    "_id" : 5,
    "name" : {
        "first" : "Ole-Johan",
        "last" : "Dahl"
    },
    "birth" : ISODate("1931-10-12T04:00:00Z"),
    "death" : ISODate("2002-06-29T04:00:00Z"),
    "contribs" : [
        "OOP",
        "Simula"
    ],
    "awards" : [
        {
            "award" : "Rosing Prize",
            "year" : 1999,
            "by" : "Norwegian Data Association"
        },
        {
```

```
            "award" : "Turing Award",
            "year" : 2001,
            "by" : "ACM"
        },
        {
            "award" : "IEEE John von Neumann Medal",
            "year" : 2001,
            "by" : "IEEE"
        }
    ]
},
{
    "_id" : 6,
    "name" : {
        "first" : "Guido",
        "last" : "van Rossum"
    },
    "birth" : ISODate("1956-01-31T05:00:00Z"),
    "contribs" : [
        "Python"
    ],
    "awards" : [
        {
            "award" : "Award for the Advancement of Free Software",
            "year" : 2001,
            "by" : "Free Software Foundation"
        },
        {
            "award" : "NLUUG Award",
            "year" : 2003,
            "by" : "NLUUG"
        }
    ]
},
{
    "_id" : ObjectId("51e062189c6ae665454e301d"),
    "name" : {
        "first" : "Dennis",
        "last" : "Ritchie"
    },
    "birth" : ISODate("1941-09-09T04:00:00Z"),
    "death" : ISODate("2011-10-12T04:00:00Z"),
    "contribs" : [
        "UNIX",
        "C"
    ],
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1983,
            "by" : "ACM"
        },
```

```
      {
        "award" : "National Medal of Technology",
        "year" : 1998,
        "by" : "United States"
      },
      {
        "award" : "Japan Prize",
        "year" : 2011,
        "by" : "The Japan Prize Foundation"
      }
    ]
},
{
    "_id" : 8,
    "name" : {
      "first" : "Yukihiro",
      "aka" : "Matz",
      "last" : "Matsumoto"
    },
    "birth" : ISODate("1965-04-14T04:00:00Z"),
    "contribs" : [
      "Ruby"
    ],
    "awards" : [
      {
        "award" : "Award for the Advancement of Free Software",
        "year" : "2011",
        "by" : "Free Software Foundation"
      }
    ]
},
{
    "_id" : 9,
    "name" : {
      "first" : "James",
      "last" : "Gosling"
    },
    "birth" : ISODate("1955-05-19T04:00:00Z"),
    "contribs" : [
      "Java"
    ],
    "awards" : [
      {
        "award" : "The Economist Innovation Award",
        "year" : 2002,
        "by" : "The Economist"
      },
      {
        "award" : "Officer of the Order of Canada",
        "year" : 2007,
        "by" : "Canada"
      }
```

```
    ]
  },
  {
    "_id" : 10,
    "name" : {
      "first" : "Martin",
      "last" : "Odersky"
    },
    "contribs" : [
      "Scala"
    ]
  }

] )


/*****************************************************************************************

-------Query# 1.1--------
Write a CRUD operation(s) that inserts the following new records into the collection
*****************************************************************************************
/


db.test.insert({
 "_id" : 20,
 "name" : {
 "first" : "Alex",
 "last" : "Chen"
 },
 "birth" : ISODate("1933-08-27T04:00:00Z"),
 "death" : ISODate("1984-11-07T04:00:00Z"),
 "contribs" : [
 "C++",
 "Simula"
 ],
 "awards" : [
 {
 "award" : "WPI Award",
 "year" : 1977,
 "by" : "WPI"
 }
 ]
})

db.test.insert({
 "_id" : 30,
 "name" : {
 "first" : "David",
 "last" : "Mark"
 },
 "birth" : ISODate("1911-04-12T04:00:00Z"),
```

```
 "death" : ISODate("2000-11-07T04:00:00Z"),
 "contribs" : [
 "C++",
 "FP",
 "Lisp",
 ],
 "awards" : [
 {
 "award" : "WPI Award",
 "year" : 1963,
 "by" : "WPI"
 },
 {
 "award" : "Turing Award",
 "year" : 1966,
 "by" : "ACM"
 }
 ]
}
);



/**********************************************************************************************

-------Query# 1.2--------
Report all documents of people who got less than 3 awards or have contribution in “FP”
**********************************************************************************************


db.test.find( { $or: [ {awards : {$exists:true}, $where:'this.awards.length<3'}, {contribs: { $in: [ 'FP']} }] } );



/**********************************************************************************************

-------Query# 1.3--------
Update the document of “Guido van Rossum” to add “OOP” to the contribution list.
**********************************************************************************************


db.test.update( { $and: [ { 'name.first' : "Guido" },{'name.last' : "van Rossum" }] } , { $push: { contribs: "OOPS" } }
;



/**********************************************************************************************

-------Query# 1.4--------
Insert a new filed of type array, called “comments”, into the document of “Alex Chen” storing the
following comments: “He taught in 3 universities”, “died from cancer”, “lived in CA”
**********************************************************************************************
```

```
db.test.update(
 { $and : [ {'name.first' : "Alex"}, {'name.last' : "Chen"} ] },
 {$set: {comments : ["He taught in 3 universities", "died from cancer", "lived in CA"]}}
);
```

/**

-------Query# 1.5--------
For each contribution by “Alex Chen”, say X, list the peoples’ names (first and last) who have contribution X. E.g., Alex Chen has two contributions in “C++” and “Simula”. Then, the output should be similar to:
a. {Contribution: “C++”,
People: [{first: “Alex”, last: “Chen”}, {first: “David”, last: “Mark”}]},
{ Contribution: “Simula”,
....}
****

```
var contri = db.test.findOne({ $and : [{'name.first' : "Alex"}, {'name.last' : "Chen"}]}).contribs;

contri.forEach(( c ) => {
 var names = db.test.find({ contribs: c }, { name: 1, _id: 0 }).toArray();
 printjson({ Contribution: c, People: names});
}
);
```

/**

-------Query# 1.6--------
Report the distinct organization that gave awards. This information can be found in the “by” field inside the “awards” array. The output should be an array of the distinct values, e.g., [“wpi’, “acm’, ...]
****

```
db.test.distinct("awards.by")
```

/**

-------Query# 1.7--------
Delete from all documents any award given on 2011.
****

```
db.test.update({
 awards: { $elemMatch: { year: 2011 } }
},
{
 $pull:
  { awards: { year: 2011 } }
}
);
```

/**

-------Query# 1.8--------
Report only the names (first and last) of those individuals who won at least two awards in 2001.
****

```
db.test.find(
{
 awards: {$exists:true},
 $where: 'this.awards.filter((a) => a.year===2001).length>1' },
 {'name.first' : 1, 'name.last' : 1}
);
```

/**

-------Query# 1.9--------
Report the document with the largest id. First, you need to find the largest _id (using a CRUD
statement), and then use that to report the corresponding document.
****

```
db.test.find().sort(
 {_id:-1}
).limit(1).pretty();
```

/**

-------Query# 1.10--------
Report only one document where one of the awards is given by “ACM”.
****

```
db.test.findOne(
```

```
 {"awards.by":"ACM"}
);



/*****************************************************************************************

-------Query# 2.1--------
Write an aggregation query that group by the award name, i.e., the “award” field inside the “awards”
array, and reports the count of each award.
*****************************************************************************************
/

db.test.aggregate([
 { $unwind: "$awards" },
 {
  $group: {
   _id : "$awards.award",
   count: { $sum: 1 }
   }
 }
])



/*****************************************************************************************

-------Query# 2.2--------
Write an aggregation query that groups by the birth year, i.e., the year within the “birth”
field, are report an array of _ids for each birth year.
/*****************************************************************************************


db.test.aggregate([
{
 $group:{_id: {$year: "$birth"},
 ids_array:{$addToSet: "$_id"}}
}
])



/*****************************************************************************************

-------Query# 2.3--------
Report the document with the smallest and largest _ids. You first need to find
the values of the smallest and largest, and then report their documents.
*****************************************************************************************
```

```
db.test.find().sort({_id: 1}).limit(1).pretty();
db.test.find().sort({_id: -1}).limit(1).pretty();
```

/**

-------Query# 3.1--------
Assume we model the records and relationships in Figure 1 using the Parent-Referencing model (Slide 4 in MongoDB-3). Write a query to report the ancestors of “MongoDB”. The output should be an array containing values [{Name: “Databases”, Level: 1},{Name: “Programming”, Level: 2}, {Name: “Books”, Level: 3}]
****

```
db.categories.insert({_id:"MongoDB",parent:"Databases"})
db.categories.insert({_id:"dbm",parent:"Databases"})
db.categories.insert({_id:"Databases",parent:"Programming"})
db.categories.insert({_id:"Languages",parent:"Programming"})
db.categories.insert({_id:"Programming",parent:"Books"})
db.categories.insert({_id:"Books",parent:null})

var allparents = [];
var stack = [];
var item = db.categories.findOne({_id: "MongoDB"});
stack.push(item);

var i=0;
while (stack.length > 0) {
 var current = stack.pop();
 i = i+1;
 if(current.parent != null){
  var par =  current.parent;
  var paritem = db.categories.findOne({_id: par});
  var val = {Name:par,Level:i};
  allparents.push(val);
  stack.push(paritem);
 }
}
allparents;
```

/**

-------Query# 3.2--------
Assume we model the records and relationships in Figure 1 using the Parent-Referencing model (Slide 4 in MongoDB-3). You are given only the root node, i.e., _id = “Books”, write a query

that reports the height of the tree. (It should be 4 in our case).
****

```
var item = db.categories.findOne({_id: "Books"});
var tree_height = 1
var stack = [];
stack.push(item);
while (stack.length > 0) {
 var current = stack.pop();
       var children =  db.categories.find( {parent: current._id});
 if (children.hasNext()) {
  var child = children.next();
  tree_height=tree_height+1
  stack.push(child);
 }
}
tree_height;
```

/**********

-------Query# 3.3--------
Assume we model the records and relationships in Figure 1 using the Child-Referencing model
(Slide 9 in MongoDB-3). Write a query to report the parent of “dbm”.
****

```
var par =  db.categories.findOne({_id: "dbm"}).parent;
```

/**********

-------Query# 3.4--------
Assume we model the records and relationships in Figure 1 using the Child-Referencing model
(Slide 9 in MongoDB-3). Write a query to report the descendants of “Books”. The output should
be an array containing values [“Programming”, “Languages”, “Databases”, “MongoDB”, “dbm”]
****

```
var descendants = [];
var stack = [];
var item = db.categories.findOne({_id: "Books"});
stack.push(item);
while (stack.length > 0) {
 var current = stack.pop();
       var children =  db.categories.find( {parent: current._id});
 while (children.hasNext() == true) {
  var child = children.next();
```

```
  descendants.push(child._id);
  stack.push(child);
 }
}

descendants;
```