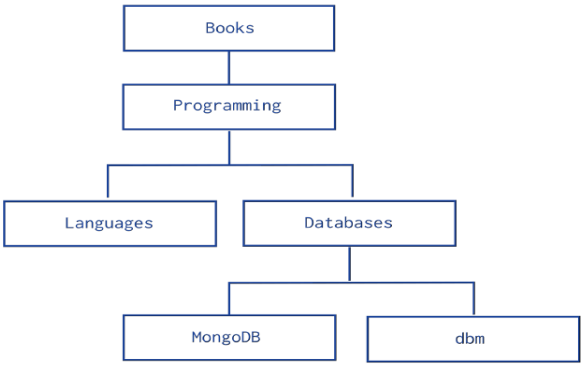
RISHITHA KIRAN, SARANYA MANOHARAN
4/14/17

BIG DATA MANAGEMENT

TEAM 24 - MONGODB

QUESTION 1



1) **Assume we model the records and relationships in Figure 1 using the Parent-Referencing model (Slide 49 in MongoDB 2). 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}]**

CREATE THE TREE:

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

QUERY:

```
var ancestors = [];
var stack = [];
var level = 1;
var item = db.categories.findOne({_id: "MongoDB"});
stack.push(item);
while (stack.length > 0) {
        var current = stack.pop();
        var parent =  db.categories.find({_id:current.parent});
                while(parent.hasNext())
                {
                        var parent_new = parent.next();
                        var result = {Name: parent_new._id, Level: level};
                        ancestors.push(result);
                        stack.push(parent_new);
                }
                level++;
}

ancestors;
```

BIG DATA MANAGEMENT

OUTPUT:

```
> db.categories.insert({_id: "MongoDB", parent:"Databases"})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "dbm", parent:"Databases"})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "Languages", parent:"Programming"})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "Databases", parent:"Programming"})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "Programming", parent:"Books"})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "Books", parent:null})
WriteResult({ "nInserted" : 1 })
```

QUERY 1 OUTPUT:

```
[
  {
        "Name" : "Databases",
        "Level" : 1
  },
  {
        "Name" : "Programming",
        "Level" : 2
  },
  {
        "Name" : "Books",
        "Level" : 3
  }
]
```

---

2) Assume we model the records and relationships in Figure 1 using the Parent-Referencing model (Slide 49 in MongoDB-2). 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 stack = [];
var item = db.categories.findOne({_id: "Books"});
stack.push(item);
var height = 0;
while (stack.length > 0){
        height++;
        var len = stack.length;
        for(var i = 0 ; i < len ; i++)
        {
                var current = stack.pop();
                var children =  db.categories.find( {parent: current._id});
                while (children.hasNext())
```

```
                {
                        var child = children.next();
                        stack.push(child);
                }
        }
}
height;
```

OUTPUT:
4

---

3) Assume we model the records and relationships in Figure 1 using the Child-Referencing model (Slide 54 in MongoDB-2). Write a query to report the parent of "dbm".

```
db.categories.remove({});
```

OUTPUT:
```
> db.categories.remove({});
WriteResult({ "nRemoved" : 6 })
```

CREATE THE TREE:
```
db.categories.insert({_id: "MongoDB", children: []})
db.categories.insert({_id: "dbm", children: []})
db.categories.insert({_id: "Languages", children: []})
db.categories.insert({_id: "Databases", children: ["MongoDB", "dbm"]})
db.categories.insert({_id: "Programming", children: ["Databases", "Languages"]})
db.categories.insert({_id: "Books",children: ["Programming"]})
```

OUTPUT:
```
> db.categories.insert({_id: "MongoDB", children: []})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "dbm", children: []})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "Languages", children: []})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "Databases", children: ["MongoDB", "dbm"]})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "Programming", children: ["Databases", "Languages"]})
WriteResult({ "nInserted" : 1 })
> db.categories.insert({_id: "Books",children: ["Programming"]})
WriteResult({ "nInserted" : 1 })
```

QUERY:
```
var parents = db.categories.find({"children" : "dbm"},{"children":0});
var parent_new = parents.next()._id;
parent_new;
```

OUTPUT:
Databases

---

4) Assume we model the records and relationships in Figure 1 using the Child-Referencing model (Slide 54 in MongoDB-2). Write a query to report the descendants of "Books". The output should be an array containing values ["Programming", "Languages", "Databases", "MongoDB", "dbm"]

```
var stack = [];
var item = db.categories.findOne({_id : "Books"});
stack.push(item);
var descendants = [];

while (stack.length > 0) {
        var length = stack.length;
        for (var i = 0; i < length; i++) {
                var current = stack.pop();
                var children = current.children;
                children.forEach(function(child){
                        db.categories.find({_id : child}).forEach(function(item) {
                                stack.push(item);
                                descendants.push(item._id);
                        });
                })
        }
}
descendants;
```

OUTPUT:

```
> descendants;
[ "Programming", "Databases", "Languages", "MongoDB", "dbm" ]
```

---

5) Assume we model the records and relationships in Figure 1 using the Child-Referencing model (Slide 54 in MongoDB-2). Write a query to report the siblings "Databases".

QUERY TYPE1:

```
var siblings = db.categories.findOne({"children" : "Databases"}).children;
for (var i = 0; i < siblings.length; i++) {
        if (siblings[i] != "Databases") {
                db.categories.findOne({_id : siblings[i]});
        }
}
```

OUTPUT TYPE1:

```
{ "_id" : "Languages", "children" : [ ] }
```

QUERY TYPE2:

```
var parent = db.categories.findOne({"children" : "Databases"});
var siblings= parent.children;
```

```
for (var i = 0; i < siblings.length; i++) {
        if (siblings[i] != "Databases") {
                db.categories.find({_id : siblings[i]},{ children :0});
        }
}
```

OUTPUT TYPE2:

```
{ "_id" : "Languages" }
```

QUESTION 2

As you did in MongoDB Project 1, Create a collection named "test", and insert into this collection the documents found in this link (10 documents): http://docs.mongodb.org/manual/reference/bios-example-collection/

1) **Write an aggregation query that groups by the award name, i.e., the "award" field inside the "awards" array, and reports the count of each award. (Use Map-Reduce mechanism)**

```
db.test.mapReduce(
        function() {
                for (var i = 0; i < this.awards.length; i++) {
                        var key = this.awards[i].award;
                        var value = 1;
                        emit(key, value);
                }
        },
        function(key, values) {
                return Array.sum(values)
        },
        {
                query : { awards : {$exists : true} },
                out : "award_count",
        }
).find()
```

OUTPUT:

```
{ "_id" : "Award for the Advancement of Free Software", "value" : 2 }
{ "_id" : "Computer Sciences Man of the Year", "value" : 1 }
{ "_id" : "Distinguished Fellow", "value" : 1 }
{ "_id" : "Draper Prize", "value" : 1 }
{ "_id" : "IEEE John von Neumann Medal", "value" : 2 }
{ "_id" : "Japan Prize", "value" : 1 }
{ "_id" : "Kyoto Prize", "value" : 1 }
{ "_id" : "NLUUG Award", "value" : 1 }
{ "_id" : "National Medal of Science", "value" : 2 }
{ "_id" : "National Medal of Technology", "value" : 2 }
{ "_id" : "Officer of the Order of Canada", "value" : 1 }
{ "_id" : "Rosing Prize", "value" : 2 }
{ "_id" : "The Economist Innovation Award", "value" : 1 }
{ "_id" : "Turing Award", "value" : 5 }
```

```
{ "_id" : "W. W. McDowell Award", "value" : 1 }
{ "_id" : "W.W. McDowell Award", "value" : 1 }
```

---

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. (Use Aggregate mechanism)

```
var result = db.test.find();
while(result.hasNext())
{
        var record = result.next();
        if (record.birth)
        {
                if(record.birth !=  null)
                {
                        record.year = record.birth.getFullYear();
                        db.temp2.insert(record);
                }
        }
}
db.temp2.aggregate ([
{
        $group: { _id: "$year", arrayIDs : { $push : "$_id" } }
} ]
);
```

OUTPUT:

```
{ "_id" : 1955, "arrayIDs" : [ 9 ] }
{ "_id" : 1965, "arrayIDs" : [ 8 ] }
{ "_id" : 1941, "arrayIDs" : [ ObjectId("51e062189c6ae665454e301d") ] }
{ "_id" : 1956, "arrayIDs" : [ 6 ] }
{ "_id" : 1931, "arrayIDs" : [ 5 ] }
{ "_id" : 1926, "arrayIDs" : [ 4 ] }
{ "_id" : 1924, "arrayIDs" : [ 1 ] }
{ "_id" : 1906, "arrayIDs" : [ 3 ] }
{ "_id" : 1927, "arrayIDs" : [ ObjectId("51df07b094c6acd67e492f41") ] }
```

---

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.

```
var large = db.test.find({}).sort({_id:-1}).limit(1);
var largestId = large.next()._id;
var small = db.test.find({}).sort({_id:1}).limit(1);
var smallestId = small.next()._id;
db.test.find({_id: {$in: [largestId, smallestId]}});
```

OUTPUT:

{ "_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("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" }] }

---

4) Use the $text operator to search for and report all documents containing "Turing Award" as one sentence (not separate keywords).

```
db.test.createIndex({ "awards.award": "text" })
db.test.find( { $text: { $search: "\"Turing Award\"" } } ).pretty();
```

OUTPUT:

{ "_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" : 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" : 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" : 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" : 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" }] }

---

5) Use the $text operator to search for and report all documents containing either "Turing" or "National Medal".

```
db.test.find({$text: { $search: "Turing National Medal"}});
```

OUTPUT:

{ "_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" : 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" : 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" : 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" : 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" }] }