Group Members: Wenhan Ji, Omkar Kulkarni

Work Distribution and Methodology:

Wenhan - 50%
1. Planning queries
2. Writing Queries
3. Writing report

Omkar - 50%
1. Panning queries
2. Writing Queries
3. Writing report

Relative Contributions: Collaborated equally on all the solutions, discussed the solution and submitted the ones which we deemed as best ones.

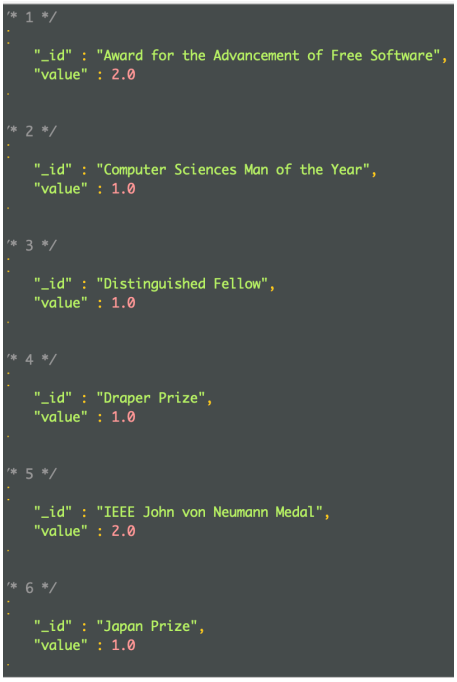
Solution:
We did the queries on **Robo 3T,** which is open source compiler for MongoDB.

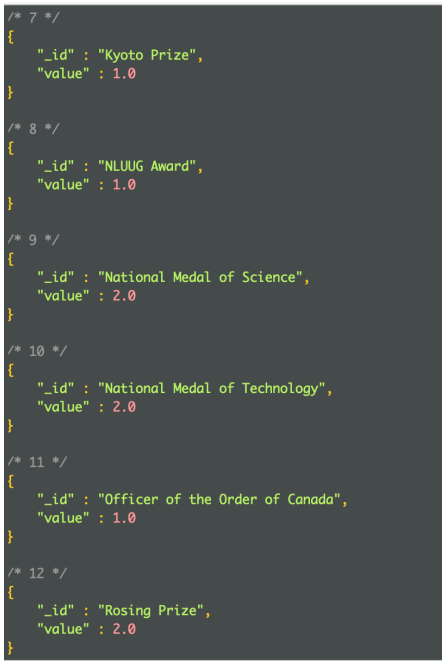
For Creating the collection, we use createcollection() with collection name in it. Then used insertMany() to insert multiple documents in it.

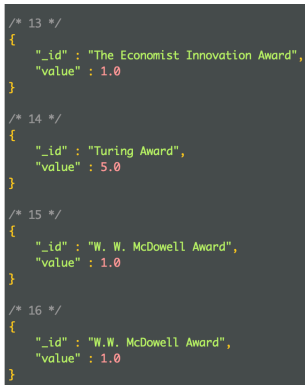
Problem 1: Query Specification over Unstructured Data in MongoDB

1) In map function, using this.award and emiting the award with 1 to count. In reduce phase, just summing up the awards by their names.

O/P:

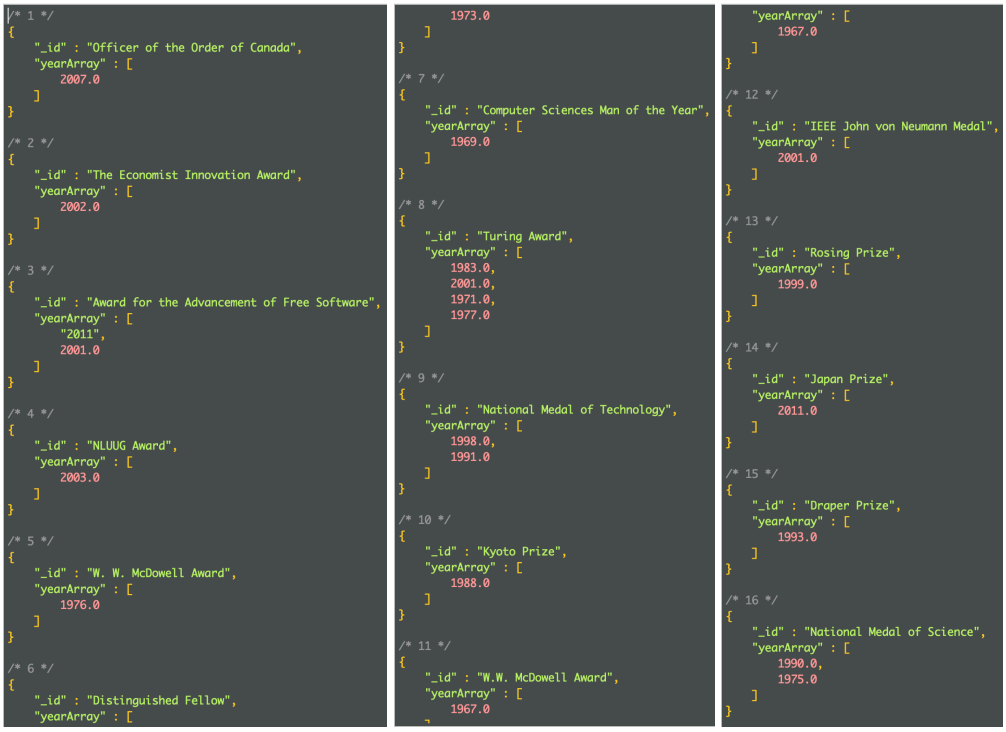






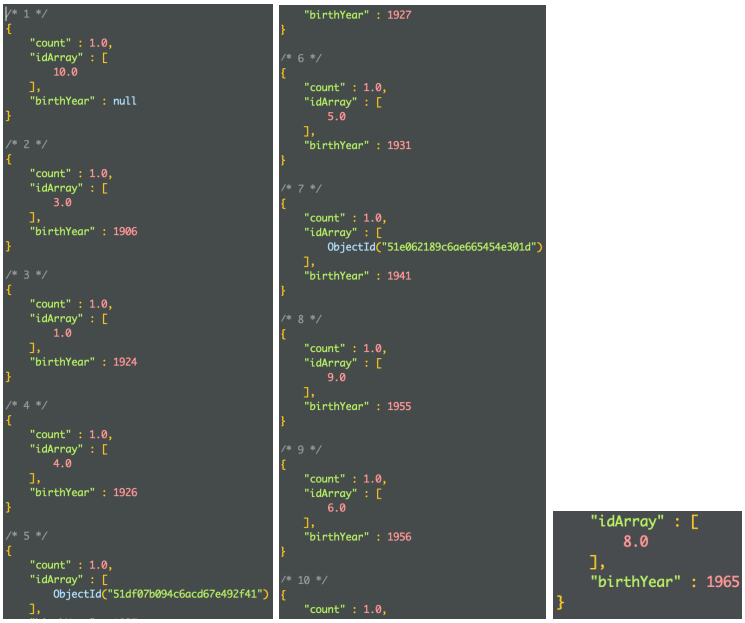
2) Here we are using aggregate function to aggregate with unwind function and then group by year array.

O/P:



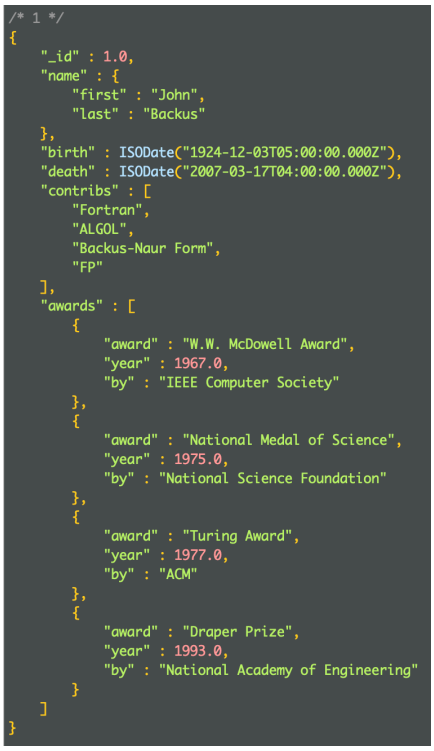
3) Again, we used the aggregate function to group by birth year, counting the number of people in it and then also displaying ids of people having same birth year

O/P:



4) We created minmaxID variable array which will hold the min and max values and then aggregate the min and max values and push them onto the array.

O/P:

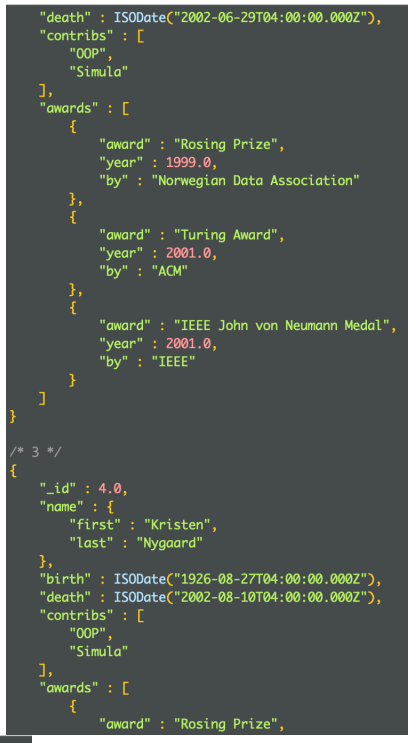


```
/* 1 */
{
    "_id" : 1.0,
    "name" : {
        "first" : "John",
        "last" : "Backus"
    },
    "birth" : ISODate("1924-12-03T05:00:00.000Z"),
    "death" : ISODate("2007-03-17T04:00:00.000Z"),
    "contribs" : [
        "Fortran",
        "ALGOL",
        "Backus-Naur Form",
        "FP"
    ],
    "awards" : [
        {
            "award" : "W.W. McDowell Award",
            "year" : 1967.0,
            "by" : "IEEE Computer Society"
        },
        {
            "award" : "National Medal of Science",
            "year" : 1975.0,
            "by" : "National Science Foundation"
        },
        {
            "award" : "Turing Award",
            "year" : 1977.0,
            "by" : "ACM"
        },
        {
            "award" : "Draper Prize",
            "year" : 1993.0,
            "by" : "National Academy of Engineering"
        }
    ]
}
```

```
/* 2 */
{
    "_id" : ObjectId("51e062189c6ae665454e301d"),
    "name" : {
        "first" : "Dennis",
        "last" : "Ritchie"
    },
    "birth" : ISODate("1941-09-09T04:00:00.000Z"),
    "death" : ISODate("2011-10-12T04:00:00.000Z"),
    "contribs" : [
        "UNIX",
        "C"
    ],
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1983.0,
            "by" : "ACM"
        },
        {
            "award" : "National Medal of Technology",
            "year" : 1998.0,
            "by" : "United States"
        },
        {
            "award" : "Japan Prize",
            "year" : 2011.0,
            "by" : "The Japan Prize Foundation"
        }
    ]
}
```

5) used ensureIndex to create text index and find to search the string and later drop the index
O/P:

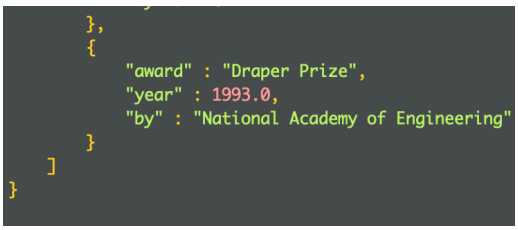
```
/* 1 */
{
    "_id" : ObjectId("51e062189c6ae665454e301d"),
    "name" : {
        "first" : "Dennis",
        "last" : "Ritchie"
    },
    "birth" : ISODate("1941-09-09T04:00:00.000Z"),
    "death" : ISODate("2011-10-12T04:00:00.000Z"),
    "contribs" : [
        "UNIX",
        "C"
    ],
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1983.0,
            "by" : "ACM"
        },
        {
            "award" : "National Medal of Technology",
            "year" : 1998.0,
            "by" : "United States"
        },
        {
            "award" : "Japan Prize",
            "year" : 2011.0,
            "by" : "The Japan Prize Foundation"
        }
    ]
}

/* 2 */
{
    "_id" : 5.0,
    "name" : {
        "first" : "Ole-Johan",
        "last" : "Dahl"
    },
    "birth" : ISODate("1931-10-12T04:00:00.000Z"),
```



```
    "death" : ISODate("2002-06-29T04:00:00.000Z"),
    "contribs" : [
        "OOP",
        "Simula"
    ],
    "awards" : [
        {
            "award" : "Rosing Prize",
            "year" : 1999.0,
            "by" : "Norwegian Data Association"
        },
        {
            "award" : "Turing Award",
            "year" : 2001.0,
            "by" : "ACM"
        },
        {
            "award" : "IEEE John von Neumann Medal",
            "year" : 2001.0,
            "by" : "IEEE"
        }
    ]
}

/* 3 */
{
    "_id" : 4.0,
    "name" : {
        "first" : "Kristen",
        "last" : "Nygaard"
    },
    "birth" : ISODate("1926-08-27T04:00:00.000Z"),
    "death" : ISODate("2002-08-10T04:00:00.000Z"),
    "contribs" : [
        "OOP",
        "Simula"
    ],
    "awards" : [
        {
            "award" : "Rosing Prize",
```

```
            "year" : 1999.0,
            "by" : "Norwegian Data Association"
        },
        {
            "award" : "Turing Award",
            "year" : 2001.0,
            "by" : "ACM"
        },
        {
            "award" : "IEEE John von Neumann Medal",
            "year" : 2001.0,
            "by" : "IEEE"
        }
    ]
}

/* 4 */
{
    "_id" : ObjectId("51df07b094c6acd67e492f41"),
    "name" : {
        "first" : "John",
        "last" : "McCarthy"
    },
    "birth" : ISODate("1927-09-04T04:00:00.000Z"),
    "death" : ISODate("2011-12-24T05:00:00.000Z"),
    "contribs" : [
        "Lisp",
        "Artificial Intelligence",
        "ALGOL"
    ],
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1971.0,
            "by" : "ACM"
        },
        {
            "award" : "Kyoto Prize",
            "year" : 1988.0,
            "by" : "Inamori Foundation"
```

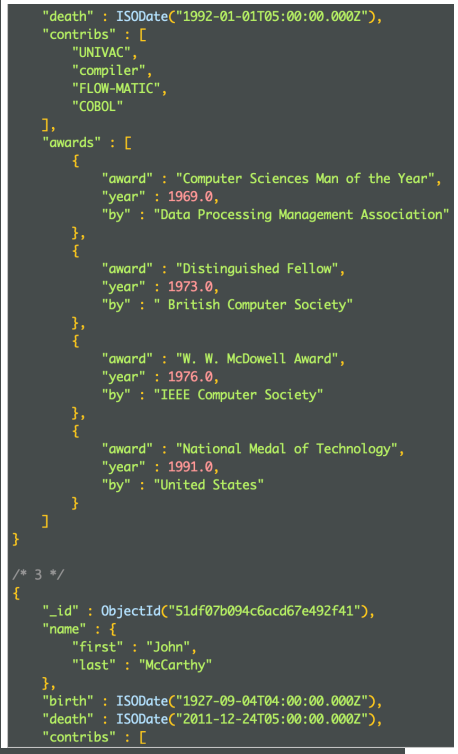
```
        },
        {
            "award" : "National Medal of Science",
            "year" : 1990.0,
            "by" : "National Science Foundation"
        }
    ]
}

/* 5 */
{
    "_id" : 1.0,
    "name" : {
        "first" : "John",
        "last" : "Backus"
    },
    "birth" : ISODate("1924-12-03T05:00:00.000Z"),
    "death" : ISODate("2007-03-17T04:00:00.000Z"),
    "contribs" : [
        "Fortran",
        "ALGOL",
        "Backus-Naur Form",
        "FP"
    ],
    "awards" : [
        {
            "award" : "W.W. McDowell Award",
            "year" : 1967.0,
            "by" : "IEEE Computer Society"
        },
        {
            "award" : "National Medal of Science",
            "year" : 1975.0,
            "by" : "National Science Foundation"
        },
        {
            "award" : "Turing Award",
            "year" : 1977.0,
            "by" : "ACM"
        },
```



```
        },
        {
            "award" : "Draper Prize",
            "year" : 1993.0,
            "by" : "National Academy of Engineering"
        }
    ]
}
```

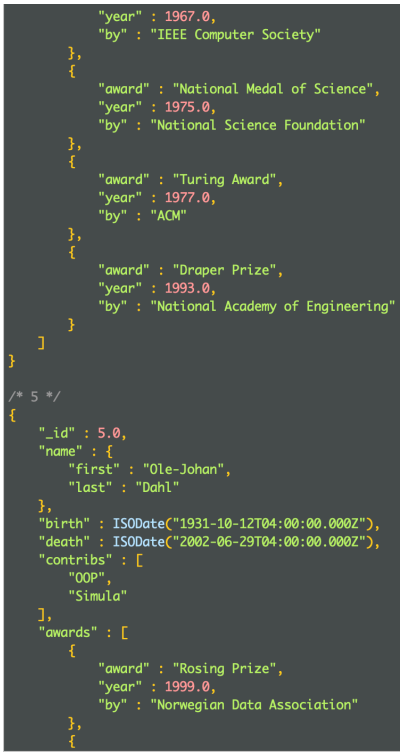
6) Same as problem 5 above just when using find() search for Turing and National Medal O/P:

```
/* 1 */
{
    "_id" : ObjectId("51e062189c6ae665454e301d"),
    "name" : {
        "first" : "Dennis",
        "last" : "Ritchie"
    },
    "birth" : ISODate("1941-09-09T04:00:00.000Z"),
    "death" : ISODate("2011-10-12T04:00:00.000Z"),
    "contribs" : [
        "UNIX",
        "C"
    ],
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1983.0,
            "by" : "ACM"
        },
        {
            "award" : "National Medal of Technology",
            "year" : 1998.0,
            "by" : "United States"
        },
        {
            "award" : "Japan Prize",
            "year" : 2011.0,
            "by" : "The Japan Prize Foundation"
        }
    ]
}

/* 2 */
{
    "_id" : 3.0,
    "name" : {
        "first" : "Grace",
        "last" : "Hopper"
    },
    "title" : "Rear Admiral",
    "birth" : ISODate("1906-12-09T05:00:00.000Z"),
```



```
    "death" : ISODate("1992-01-01T05:00:00.000Z"),
    "contribs" : [
        "UNIVAC",
        "compiler",
        "FLOW-MATIC",
        "COBOL"
    ],
    "awards" : [
        {
            "award" : "Computer Sciences Man of the Year",
            "year" : 1969.0,
            "by" : "Data Processing Management Association"
        },
        {
            "award" : "Distinguished Fellow",
            "year" : 1973.0,
            "by" : " British Computer Society"
        },
        {
            "award" : "W. W. McDowell Award",
            "year" : 1976.0,
            "by" : "IEEE Computer Society"
        },
        {
            "award" : "National Medal of Technology",
            "year" : 1991.0,
            "by" : "United States"
        }
    ]
}

/* 3 */
{
    "_id" : ObjectId("51df07b094c6acd67e492f41"),
    "name" : {
        "first" : "John",
        "last" : "McCarthy"
    },
    "birth" : ISODate("1927-09-04T04:00:00.000Z"),
    "death" : ISODate("2011-12-24T05:00:00.000Z"),
    "contribs" : [
```

```
        "Lisp",
        "Artificial Intelligence",
        "ALGOL"
    ],
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1971.0,
            "by" : "ACM"
        },
        {
            "award" : "Kyoto Prize",
            "year" : 1988.0,
            "by" : "Inamori Foundation"
        },
        {
            "award" : "National Medal of Science",
            "year" : 1990.0,
            "by" : "National Science Foundation"
        }
    ]
}

/* 4 */
{
    "_id" : 1.0,
    "name" : {
        "first" : "John",
        "last" : "Backus"
    },
    "birth" : ISODate("1924-12-03T05:00:00.000Z"),
    "death" : ISODate("2007-03-17T04:00:00.000Z"),
    "contribs" : [
        "Fortran",
        "ALGOL",
        "Backus-Naur Form",
        "FP"
    ],
    "awards" : [
        {
            "award" : "W.W. McDowell Award",
```



```
            "year" : 1967.0,
            "by" : "IEEE Computer Society"
        },
        {
            "award" : "National Medal of Science",
            "year" : 1975.0,
            "by" : "National Science Foundation"
        },
        {
            "award" : "Turing Award",
            "year" : 1977.0,
            "by" : "ACM"
        },
        {
            "award" : "Draper Prize",
            "year" : 1993.0,
            "by" : "National Academy of Engineering"
        }
    ]
}

/* 5 */
{
    "_id" : 5.0,
    "name" : {
        "first" : "Ole-Johan",
        "last" : "Dahl"
    },
    "birth" : ISODate("1931-10-12T04:00:00.000Z"),
    "death" : ISODate("2002-06-29T04:00:00.000Z"),
    "contribs" : [
        "OOP",
        "Simula"
    ],
    "awards" : [
        {
            "award" : "Rosing Prize",
            "year" : 1999.0,
            "by" : "Norwegian Data Association"
        },
        {
```

```
            "award" : "Turing Award",
            "year" : 2001.0,
            "by" : "ACM"
        },
        {
            "award" : "IEEE John von Neumann Medal",
            "year" : 2001.0,
            "by" : "IEEE"
        }
    ]
}

/* 6 */
{
    "_id" : 4.0,
    "name" : {
        "first" : "Kristen",
        "last" : "Nygaard"
    },
    "birth" : ISODate("1926-08-27T04:00:00.000Z"),
    "death" : ISODate("2002-08-10T04:00:00.000Z"),
    "contribs" : [
        "OOP",
        "Simula"
    ],
    "awards" : [
        {
            "award" : "Rosing Prize",
            "year" : 1999.0,
            "by" : "Norwegian Data Association"
        },
        {
            "award" : "Turing Award",
            "year" : 2001.0,
            "by" : "ACM"
        },
        {
            "award" : "IEEE John von Neumann Medal",
            "year" : 2001.0,
            "by" : "IEEE"
        }
```

7) same as problem 5 just when using find() use "\"Turing\" \"National Medal\"" which will give results for both awards.

O/P:

```
/* 1 */
{
    "_id" : ObjectId("51e062189c6ae665454e301d"),
    "name" : {
        "first" : "Dennis",
        "last" : "Ritchie"
    },
    "birth" : ISODate("1941-09-09T04:00:00.000Z"),
    "death" : ISODate("2011-10-12T04:00:00.000Z"),
    "contribs" : [
        "UNIX",
        "C"
    ],
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1983.0,
            "by" : "ACM"
        },
        {
            "award" : "National Medal of Technology",
            "year" : 1998.0,
            "by" : "United States"
        },
        {
            "award" : "Japan Prize",
            "year" : 2011.0,
            "by" : "The Japan Prize Foundation"
        }
    ]
}

/* 2 */
{
    "_id" : ObjectId("51df07b094c6acd67e492f41"),
    "name" : {
        "first" : "John",
        "last" : "McCarthy"
    },
    "birth" : ISODate("1927-09-04T04:00:00.000Z"),
    "death" : ISODate("2011-12-24T05:00:00.000Z"),
    "contribs" : [
        "Lisp",
        "Artificial Intelligence",
        "ALGOL"
    ],
    "awards" : [
        {
            "award" : "Turing Award",
            "year" : 1971.0,
            "by" : "ACM"
        },
        {
            "award" : "Kyoto Prize",
            "year" : 1988.0,
            "by" : "Inamori Foundation"
        },
        {
            "award" : "National Medal of Science",
            "year" : 1990.0,
            "by" : "National Science Foundation"
        }
    ]
}

/* 3 */
{
    "_id" : 1.0,
    "name" : {
        "first" : "John",
        "last" : "Backus"
    },
    "birth" : ISODate("1924-12-03T05:00:00.000Z"),
    "death" : ISODate("2007-03-17T04:00:00.000Z"),
    "contribs" : [
        "Fortran",
        "ALGOL",
        "Backus-Naur Form",
        "FP"
    ],
    "awards" : [
        {
            "award" : "W.W. McDowell Award",
            "year" : 1967.0,
            "by" : "IEEE Computer Society"
        },
        {
            "award" : "National Medal of Science",
            "year" : 1975.0,
            "by" : "National Science Foundation"
        },
        {
            "award" : "Turing Award",
            "year" : 1977.0,
            "by" : "ACM"
        },
        {
            "award" : "Draper Prize",
            "year" : 1993.0,
            "by" : "National Academy of Engineering"
        }
    ]
}
```

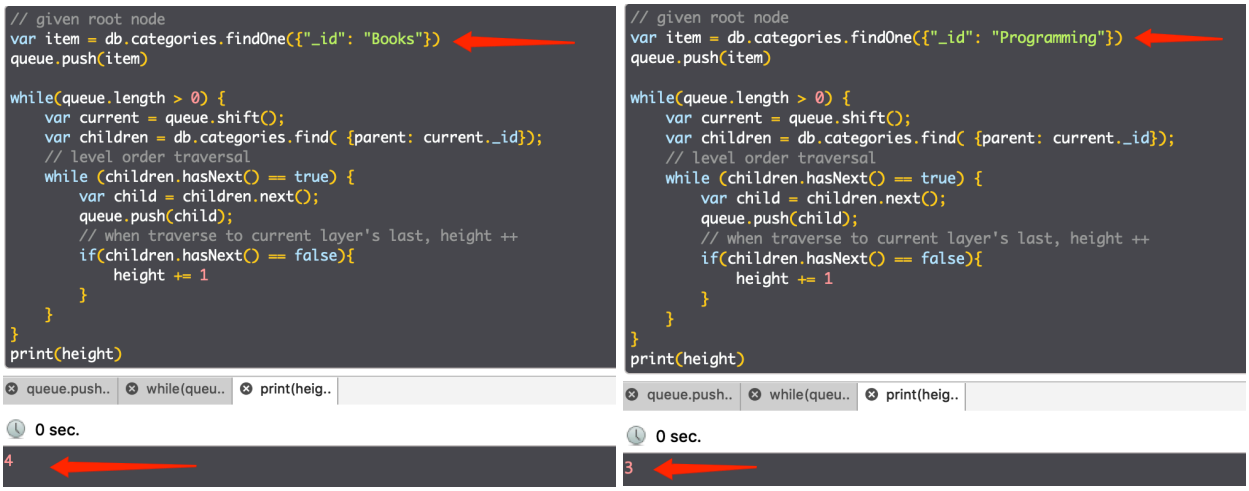
Problem 2: Querying Parent-Child Relationships in MongoDB

1-2) Model the records and relations in Figure 1 using Parent-Referencing model.
Here we create the categories collection. In this collection, each document represents one node in the tree structured relationship. Every document has 2 fields, one represents the node name and another reference to its parent node name.

```
db.createCollection("categories")

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 } )
```

1) In this problem, we use breadth first search algorithm to level-order traverse the tree.
For breadth first search, we implement it using queue. For each layer of the tree, we let height variable increment 1 when traverse to the last node of the layer. Finally, we can get height of the tree.

Below shows the result we get.
Given root node Books, we get height 4.
Given root node Programming, we get height 3.



2) We use one pointer (item variable) to iteratively points to the parent of the node until root. For each iteration, we can get one direct parent node and its level.

Below shows the ancestors of “DBM”

```
/* 1 */
[
    {
        "Name" : "Databases",
        "Level" : 1.0
    },
    {
        "Name" : "Programming",
        "Level" : 2.0
    },
    {
        "Name" : "Books",
        "Level" : 3.0
    }
]
```

3-5) Model the records and relations in Figure 1 using Child-Referencing model.
Here we drop the previous collection and create a new categories collection.
In this collection, each document represents one node in the tree structured relationship.
Every document has 2 fields, one represents the node name and another is a list of it’s children name.

```
db.categories.drop() // drop parent-referecing model
db.createCollection("categories")
db.categories.insert( { _id: "MongoDB", children: [] } )
db.categories.insert( { _id: "dbm", children: [] } )
db.categories.insert( { _id: "Databases", children: [ "MongoDB", "dbm" ] } )
db.categories.insert( { _id: "Languages", children: [] } )
db.categories.insert( { _id: "Programming", children: [ "Databases", "Languages" ] } )
db.categories.insert( { _id: "Books", children: [ "Programming" ] } )
```

3) We use find to query the node whose children has MongoDB.
The result shows the direct parent of MongoDB.

```
/* 1 */
{
    "_id" : "Databases",
    "children" : [
        "MongoDB",
        "dbm"
    ]
}
```

4) Here we use the breadth first search to level-order traverse the tree.
During traversal, we push each descendant node's name to a result list.
Given root node "Programming", we get four descendants' name.

```
/* 1 */
[
    "Databases",
    "Languages",
    "MongoDB",
    "dbm"
]
```

5) First, we find the parent of "Language" node.
Then we loop over the parent's children nodes and push the corresponding document to a result list except for "language" itself.
We can get it sibling "Databases".

```
/* 1 */
[
    {
        "_id" : "Databases",
        "children" : [
            "MongoDB",
            "dbm"
        ]
    }
]
```