Juliana Porto

April 29, 2024

DS503 Big Data Management

Project 4

Question	Status	Comment
Q1	Fully Working	**Part 1:** Write a CRUD operation(s) that inserts the following new records into the collection. *Plan*: • Use insertMany command to insert all records at once *Results:* 

```
mydb> db.test0.insertMany([
...     {
.....         "_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"
.......               }
.....         ]
.....     },
...     {
.....         "_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"
.......               }
.....         ]
.....     }
... ])
{ acknowledged: true, insertedIds: { '0': 20, '1': 30 } }
```

Part 2: Report all documents of people who got less than 3 awards or have contribution in “FP”.

Plan:

- Do an aggregation to match for values where either…
  - Size of “awards” array is less than 3
    - Or 0 (AKA “awards” does exist)
  - “FP” is in “contribs”

Results:

https://mongoplayground.net/p/-5_vc_jaeoT

Part 3: Insert a new field of type array, called “comments”, into the document of “Alex Chen” storing the following commands: “He taught me in 3 universities”, “died from cancer”, “lived in CA”.

Plan:

- Get doc of Alex Chen
- Insert 3 comments into comments array

Results:

https://mongoplayground.net/p/Hq_58SCM2RU

Part 4: 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:

{Contribution: “C++”,
People: [{first: “Alex”, last: “Chen”}, {first: “David”, last: “Mark”}] },
{Contribution: “Simula”,
…}

Plan:

- Get Alex
- Get Alex’s contribs
- Group by contrib
- Add Alex first & last name to initial person
- Get other ppl with same contribs
- Make sure to not include Alex again so there aren’t duplicates in list of ppl
- Add first & last name of other people
- Project output in "Contribution": … "People": … format for each contrib

Results:

		https://mongoplayground.net/p/5Zpg6V4Rx4M **Part 5:** 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", ...]. *Plan:* • Get all awards • Group by the "by" field to get distinct orgs • Project org names *Results:* https://mongoplayground.net/p/6fveu2NOFBw **Source Code:** Project4/bios1.js
		Additional Notes: • Code was run and tested in Mongo Playground • All code for parts 2 through 5 can be run using the included links • The Playground only supports the find(), aggregate(), update() and explain() methods, so I had to add a screenshot of part 1 run in the docker container setup instead since it uses the insertMany() method • *Note*: formatting gets a little funky with the playground links, but correctly formatted code can be found in the .js files provided
Q2	Fully Working	**Part 1:** Write an aggregation query that group the award name, i.e., the "award" field inside the "awards" array and reports the count of each award. *Plan:* • Get awards • Group by the "award" field • Count the number of each award • Project award name & count *Results:* https://mongoplayground.net/p/yVEYTo5IUnR **Part 2:** Write an aggregation query that groups by the birth year, i.e., the year within the "birth" field, and report an array of _ids for each birth year. *Plan:* • Get birth year • Group by birth year and get _ids in array • Project birth year & ids

		Results: https://mongoplayground.net/p/Bkq_nUGMPF4 **Part 3:** Write an aggregation query that groups by the award's year (which is found inside the awards array), and for each year, report the count of people who received awards in this year. *Plan:* • Get awards • Group by the " year" field • Count the number of each award in that year • Project award year & count *Results:* https://mongoplayground.net/p/Dc58CAsS60W **Source Code:** Project4/bios2.js
		Additional Notes: • Code was run and tested in Mongo Playground • All code for parts 1 through 3 can be run using the included links
Q3	Fully Working	**Setup:** Assume we model the records and relationships in Figure 1 using the Parent-Referencing model (Slide 4 in Mongo-BD-4). This model was saved as *categories*.  Assume we model the records and relationships in Figure 1 using the Child-Referencing model (Slide 8 in Mongo-BD-4). This model was saved as *categories2*.

```
mydb> db.categories2.insert({ _id: "MongoDB", children: [] })
{ acknowledged: true, insertedIds: { '0': 'MongoDB' } }
mydb> db.categories2.insert({ _id: "dbm", children: [] })
{ acknowledged: true, insertedIds: { '0': 'dbm' } }
mydb> db.categories2.insert({ _id: "Databases", children: ["MongoDB", "dbm"] })
{ acknowledged: true, insertedIds: { '0': 'Databases' } }
mydb> db.categories2.insert({ _id: "Languages", children: [] })
{ acknowledged: true, insertedIds: { '0': 'Languages' } }
mydb> db.categories2.insert({ _id: "Programming", children: ["Databases", "Languages"] })
{ acknowledged: true, insertedIds: { '0': 'Programming' } }
mydb> db.categories2.insert({ _id: "Books", children: ["Programming"] })
{ acknowledged: true, insertedIds: { '0': 'Books' } }
mydb> db.categories2.find({ })
[
  { _id: 'MongoDB', children: [] },
  { _id: 'dbm', children: [] },
  { _id: 'Databases', children: [ 'MongoDB', 'dbm' ] },
  { _id: 'Languages', children: [] },
  { _id: 'Programming', children: [ 'Databases', 'Languages' ] },
  { _id: 'Books', children: [ 'Programming' ] }
]
```

Part 1: Assume we model the records and relationships in Figure 1 using the Parent-Referencing model.

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}].

Note: “Level” is the distance from “MongoDB” node to the other node. It should be computed in the code.

Plan:

- Create a function that…
  - Takes in a leaf ID
  - Finds current based on leaf ID
  - Creates an array to store ancestors
  - Sets current level to 1
  - Goes through the tree in a while look until the root node is reached to…
    - Add current to ancestors array
    - Find the parent node of current and set that to the new current to continue moving through the tree
  - Return ancestors
- Run the function with “MongoDB” as the leaf ID
- Print the results

Results:

```
mydb> function getAncestors(leafID) {
...     // Find current based on ID
...     let current = db.categories.findOne({ _id: leafID });
...
...     // If current doesn't exist, return null
...     if (!current) {
...         return null;
...     }
...
...     // Create array to store ancestors
...     let ancestors = [];
...     // Set current level
...     let level = 1;
...
...     // Go through the tree until we reach the root
...     while (current.parent !== null) {
...         // Add current to ancestors array
...         ancestors.unshift({ Name: current.parent, Level: level++ });
...
...         // Find parent node of current to keep moving through tree
...         current = db.categories.findOne({ _id: current.parent });
...     }
...
...     // Return ancestors
...     return ancestors;
... }
[Function: getAncestors]
mydb>

mydb> // Run getAncestors with leafID = "MongoDB"

mydb> let result = getAncestors("MongoDB");

mydb> printjson(result);
[
  { Name: 'Books', Level: 3 },
  { Name: 'Programming', Level: 2 },
  { Name: 'Databases', Level: 1 }
]
```

Part 2: Assume we model the records and relationships in Figure 1 using the Parent-Referencing model.

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.

Plan:

- Create a function that…
  - Takes in a root ID
  - Finds current based on root ID
  - Creates a stack
  - Sets current height to 0
  - Pushes root to stack
  - Loops through stack while there’s still stuff in it…
    - Pops to get current
    - Finds children of current (AKA who has current as parent)
    - While more children exist, loop thorugh and…
      - Set current child to next

- Push child to stack
- Calculate height
  - Return height
- Run the function with “Books” as the root ID
- Print the results

Results:

```
mydb> function getTreeHeight(rootID) {
...     // Find root based on ID
...     var root = db.categories.findOne({ _id: rootID });
...
...     // If root doesn't exist, return null
...     if (!root) {
...         return null;
...     }
...
...     // Create stack
...     var stack = [];
...     // Set current height
...     var height = 0;
...
...     // Push root level
...     stack.push({ node: root, level: 1 });
...
...     // While there's still stuff in the stack...
...     while (stack.length > 0) {
...         // Pop to get current
...         var current = stack.pop();
...
...         // Find children of current (AKA who has current as parent)
...         var children = db.categories.find({ parent: current.node._id });
...
...         // While more children exist...
...         while (children.hasNext()) {
...             // Set current child to next child
...             var child = children.next();
...
...             // Push child to stack
...             stack.push({ node: child, level: current.level + 1 });
...
...             // Calculate height
...             height = Math.max(height, current.level + 1);
...         }
...     }
...
...     // Return height
...     return height;
... }
[Function: getTreeHeight]
mydb>

mydb> // Run getTreeHeight with rootID = "Books"

mydb> var result2 = getTreeHeight("Books");

mydb> printjson(result2);
4
```

Part 3: Assume we model the records and relationships in Figure 1 using the Child-Referencing model.

Write a query to report the descendants of “Books”. The output should be an array containing values [“Programming”, “Languages”, “Databases”, “MongoDB”, “dbm”].

Plan:

- Create a function that…
  - Takes in a root ID
  - Finds current based on root ID
  - Creates an array to store descendants
  - Creates a stack
  - Pushes root to stack
  - Loops through stack while there's still stuff in it…
    - Pops to get current
    - Finds children of current
    - While more children exist, loop through and…
      - Add children to descendants
      - Push each child to stack
  - Return descendants
- Run the function with "Books" as the root ID
- Print the results

Results:

```
mydb> function getDescendants(rootID) {
...     // Find root based on ID
...     var root = db.categories2.findOne({ _id: rootID });
...
...     // If root doesn't exist, return null
...     if (!root) {
...         return null;
...     }
...
...     // Create array to store descendants
...     var descendants = [];
...     // Create stack
...     var stack = [];
...
...     // Push root
...     stack.push(root);
...
...     // While there's still stuff in the stack...
...     while (stack.length > 0) {
...         // Pop to get current
...         var current = stack.pop();
...         // Find children of current
...         var children = db.categories2.findOne({ _id: current._id }).children;
...
...         // While more children exist...
...         if (children.length > 0) {
...             // Add children to descendants
...             descendants = descendants.concat(children);
...             // For each child...
...             children.forEach(function (child) {
.....                 // Push onto stack
.....                 stack.push({ _id: child });
.....             });
...         }
...     }
...
...     // Return descendants
...     return descendants;
... }
[Function: getDescendants]
mydb>

mydb> // Run getDescendants with rootID = "Books"

mydb> var result3 = getDescendants("Books");

mydb> printjson(result3);
[ 'Programming', 'Databases', 'Languages', 'MongoDB', 'dbm' ]
```

		Source Code: Project4/tree.js
		Additional Notes: • Code was run and tested using docker container setup