THE UNIVERSITY OF
SYDNEY

School of Information Technologies
Faculty of Engineering & IT

ASSIGNMENT/PROJECT COVERSHEET - GROUP ASSESSMENT

Unit of Study: comp5338 Advanced data models

Assignment name: Polyglot Persistence with NoSQL Systems

Tutorial time: 10/10/2017 8 PM-9 PM **Tutor name:**__________________

DECLARATION
We the undersigned declare that we have read and understood the *University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy*, an, and except where specifically acknowledged, the work contained in this assignment/project is our own work, and has not been copied from other sources or been previously submitted for award or assessment.

We understand that failure to comply with the *Academic Dishonesty and Plagiarism in Coursework Policy* can lead to severe penalties as outlined under Chapter 8 of the *University of Sydney By-Law 1999* (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

We realise that we may be asked to identify those portions of the work contributed by each of us and required to demonstrate our individual knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

Project team members				
Student name	**Student ID**	**Participated**	**Agree to share**	**Signature**
1. Mingxuan LI	470325230	[✓ Yes] / No	[✓ Yes] / No	Mingxuan Li
2. Zhiliang WANG	460094203	[✓ Yes] / No	[✓ Yes] / No	
3.		Yes / No	Yes / No	
4.		Yes / No	Yes / No	
5.		Yes / No	Yes / No	
6.		Yes / No	Yes / No	
7.		Yes / No	Yes / No	
8.		Yes / No	Yes / No	
9.		Yes / No	Yes / No	
10.		Yes / No	Yes / No	

COMP 5338 Group 08

Assignment: Polyglot Persistence with NoSQL Systems

Group members:

Mingxuan LI, 470325230

Zhiliang WANG, 460094203

1 Introduction

This report will briefly introduce the schema designs for both MongoDB and neo4j storage systems for this assignment, then followed by the query design and execution using command line tool to fetch the correct result given simple queries and analytics queries, at last, this report will compare these two storage systems with their pros and cons regarding the query performances, data schema design etc. You can also find the source code in the following Github link:

https://github.com/MingxuanLi/comp5338-polygot-persistence-systems

2 Schema Design

The following diagrams briefly describe the data model schema and their relationships in both MongoDB and neo4j.

2.1 MongoDB Schema Design

For MongoDB part, we use mongoose to explicitly design the schema in the javascript files, the schemas are under ***src/mongo-schemas***. The data model diagram looks like this:

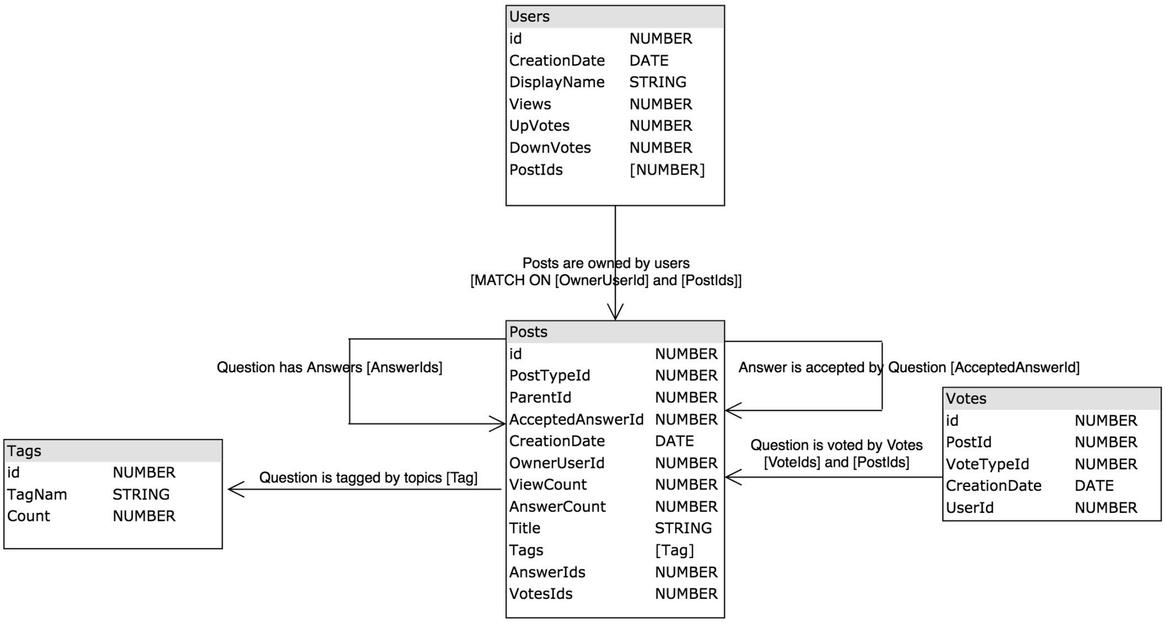
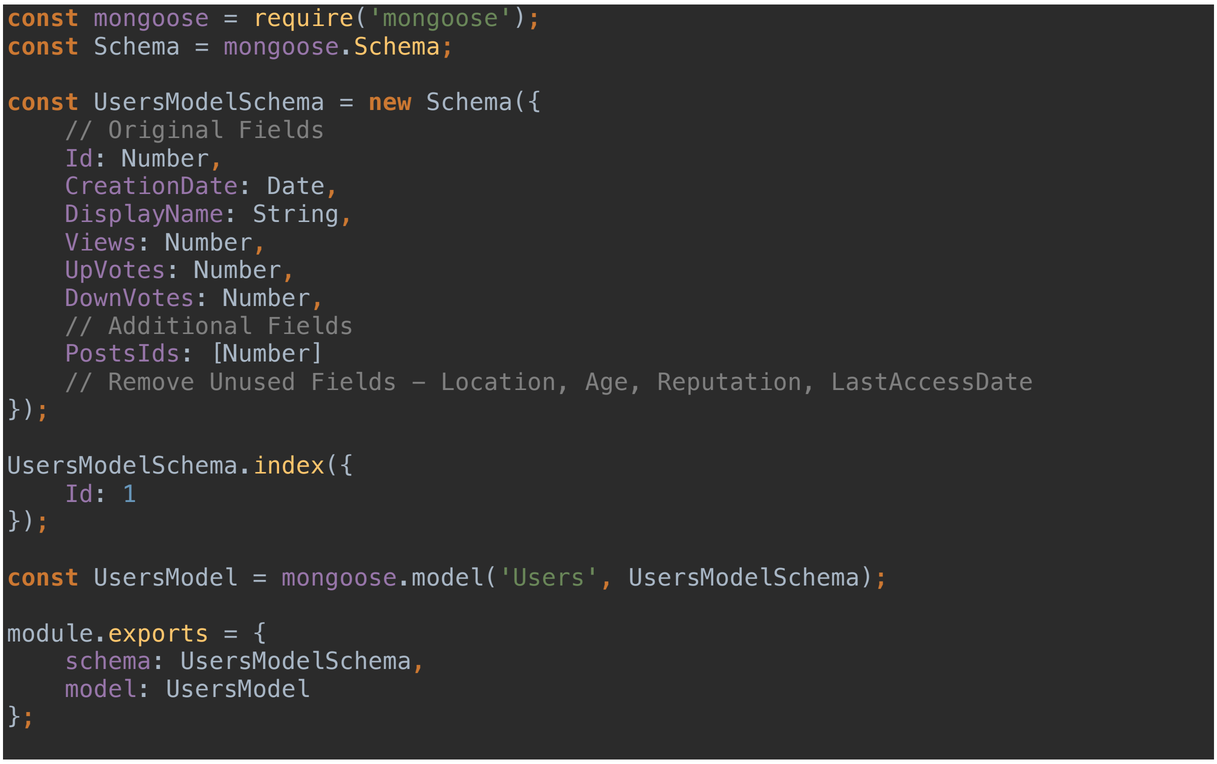


Figure 2.1 Relations in mongodb schema design

Mongodb has three representative query design method: **One-to-One Relationships with Embedded Documents, One-to-Many Relationships with Embedded Documents, One-to-Many Relationships with Document References**. In our assignment, based on these principles, we follow these rules to implement our query.

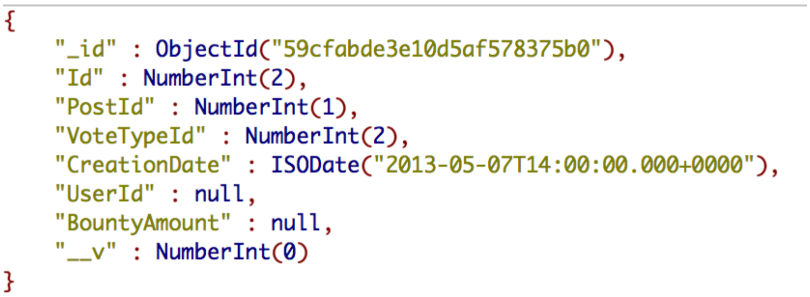
One of the example collection User looks like this with its index.



```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const UsersModelSchema = new Schema({
    // Original Fields
    Id: Number,
    CreationDate: Date,
    DisplayName: String,
    Views: Number,
    UpVotes: Number,
    DownVotes: Number,
    // Additional Fields
    PostsIds: [Number]
    // Remove Unused Fields - Location, Age, Reputation, LastAccessDate
});

UsersModelSchema.index({
    Id: 1
});

const UsersModel = mongoose.model('Users', UsersModelSchema);

module.exports = {
    schema: UsersModelSchema,
    model: UsersModel
};
```

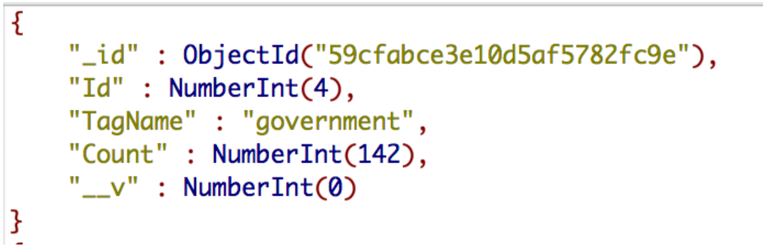
2.1.1 One-to-One Relationships with Embedded Documents

Presents a data model that uses embedded documents to describe one-to-one relationships between connected data.



```
{
    "_id" : ObjectId("59cfabde3e10d5af578375b0"),
    "Id" : NumberInt(2),
    "PostId" : NumberInt(1),
    "VoteTypeId" : NumberInt(2),
    "CreationDate" : ISODate("2013-05-07T14:00:00.000+0000"),
    "UserId" : null,
    "BountyAmount" : null,
    "__v" : NumberInt(0)
}
```

Figure 2.3 Sample document in collection **Votes**

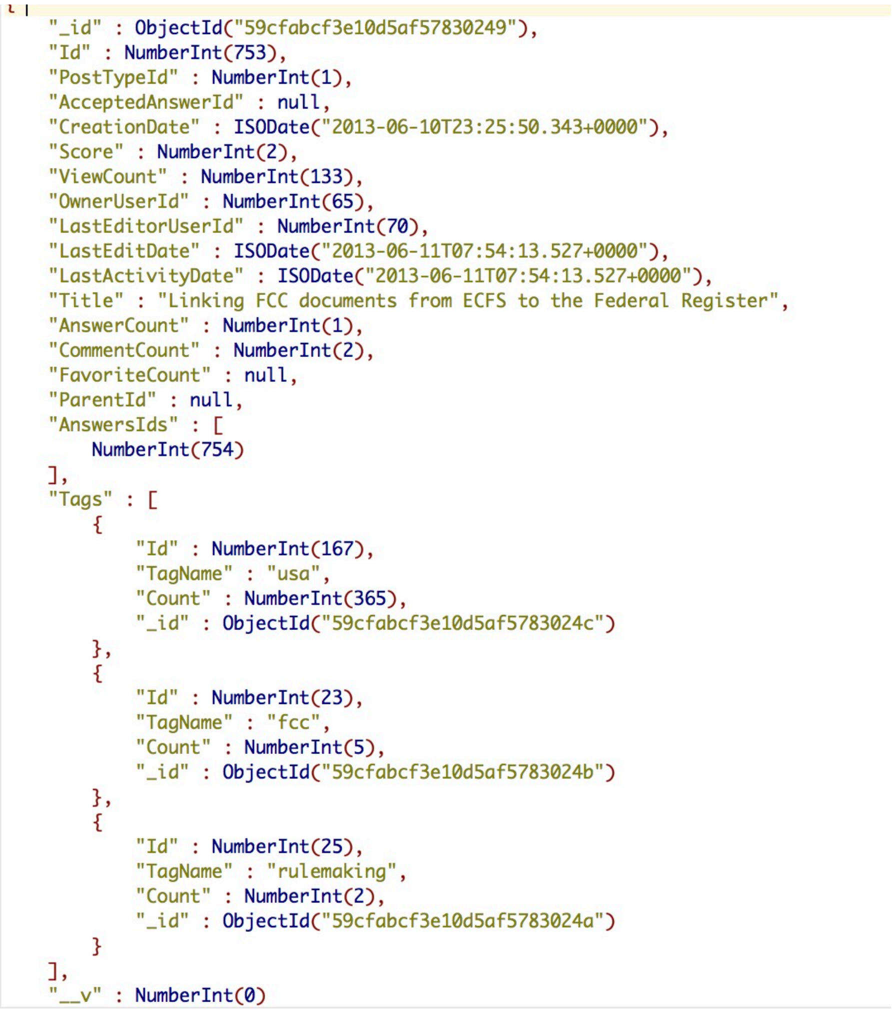


```
{
    "_id" : ObjectId("59cfabce3e10d5af5782fc9e"),
    "Id" : NumberInt(4),
    "TagName" : "government",
    "Count" : NumberInt(142),
    "__v" : NumberInt(0)
}
```

Figure 2.4 Sample document in collection **Tags**

2.1.2 One-to-Many Relationships with Embedded Documents

Presents a data model that uses embedded documents to describe one-to-many relationships between connected data.

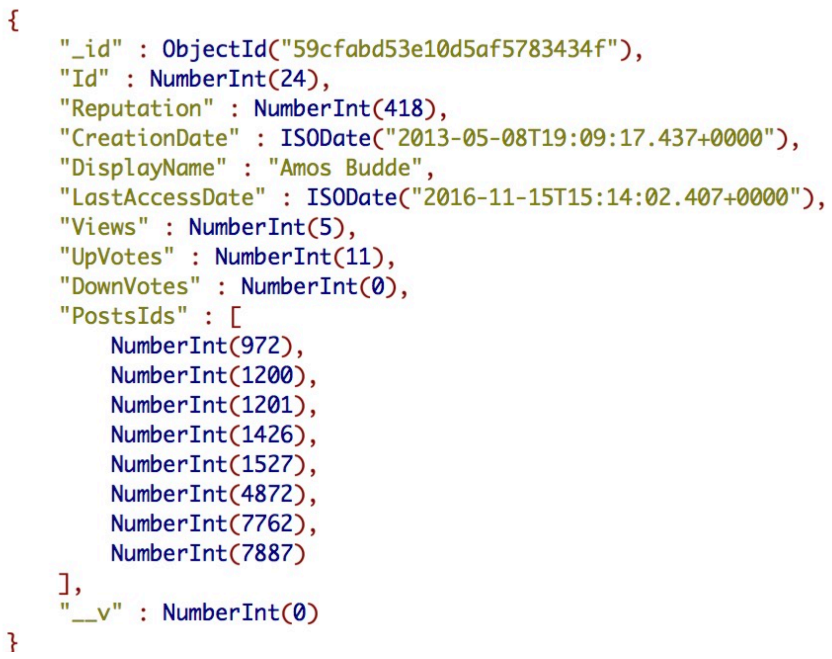


```
{
    "_id" : ObjectId("59cfabcf3e10d5af57830249"),
    "Id" : NumberInt(753),
    "PostTypeId" : NumberInt(1),
    "AcceptedAnswerId" : null,
    "CreationDate" : ISODate("2013-06-10T23:25:50.343+0000"),
    "Score" : NumberInt(2),
    "ViewCount" : NumberInt(133),
    "OwnerUserId" : NumberInt(65),
    "LastEditorUserId" : NumberInt(70),
    "LastEditDate" : ISODate("2013-06-11T07:54:13.527+0000"),
    "LastActivityDate" : ISODate("2013-06-11T07:54:13.527+0000"),
    "Title" : "Linking FCC documents from ECFS to the Federal Register",
    "AnswerCount" : NumberInt(1),
    "CommentCount" : NumberInt(2),
    "FavoriteCount" : null,
    "ParentId" : null,
    "AnswersIds" : [
        NumberInt(754)
    ],
    "Tags" : [
        {
            "Id" : NumberInt(167),
            "TagName" : "usa",
            "Count" : NumberInt(365),
            "_id" : ObjectId("59cfabcf3e10d5af5783024c")
        },
        {
            "Id" : NumberInt(23),
            "TagName" : "fcc",
            "Count" : NumberInt(5),
            "_id" : ObjectId("59cfabcf3e10d5af5783024b")
        },
        {
            "Id" : NumberInt(25),
            "TagName" : "rulemaking",
            "Count" : NumberInt(2),
            "_id" : ObjectId("59cfabcf3e10d5af5783024a")
        }
    ],
    "__v" : NumberInt(0)
```

Figure 2.5 Sample document in collection **Posts**

2.1.3 One-to-Many Relationships with Document References

Presents a data model that uses references to describe one-to-many relationships between documents.



```
{
    "_id" : ObjectId("59cfabd53e10d5af5783434f"),
    "Id" : NumberInt(24),
    "Reputation" : NumberInt(418),
    "CreationDate" : ISODate("2013-05-08T19:09:17.437+0000"),
    "DisplayName" : "Amos Budde",
    "LastAccessDate" : ISODate("2016-11-15T15:14:02.407+0000"),
    "Views" : NumberInt(5),
    "UpVotes" : NumberInt(11),
    "DownVotes" : NumberInt(0),
    "PostsIds" : [
        NumberInt(972),
        NumberInt(1200),
        NumberInt(1201),
        NumberInt(1426),
        NumberInt(1527),
        NumberInt(4872),
        NumberInt(7762),
        NumberInt(7887)
    ],
    "__v" : NumberInt(0)
}
```

Figure 2.6 Sample document in collection **Users**

2.2 neo4j Schema Design

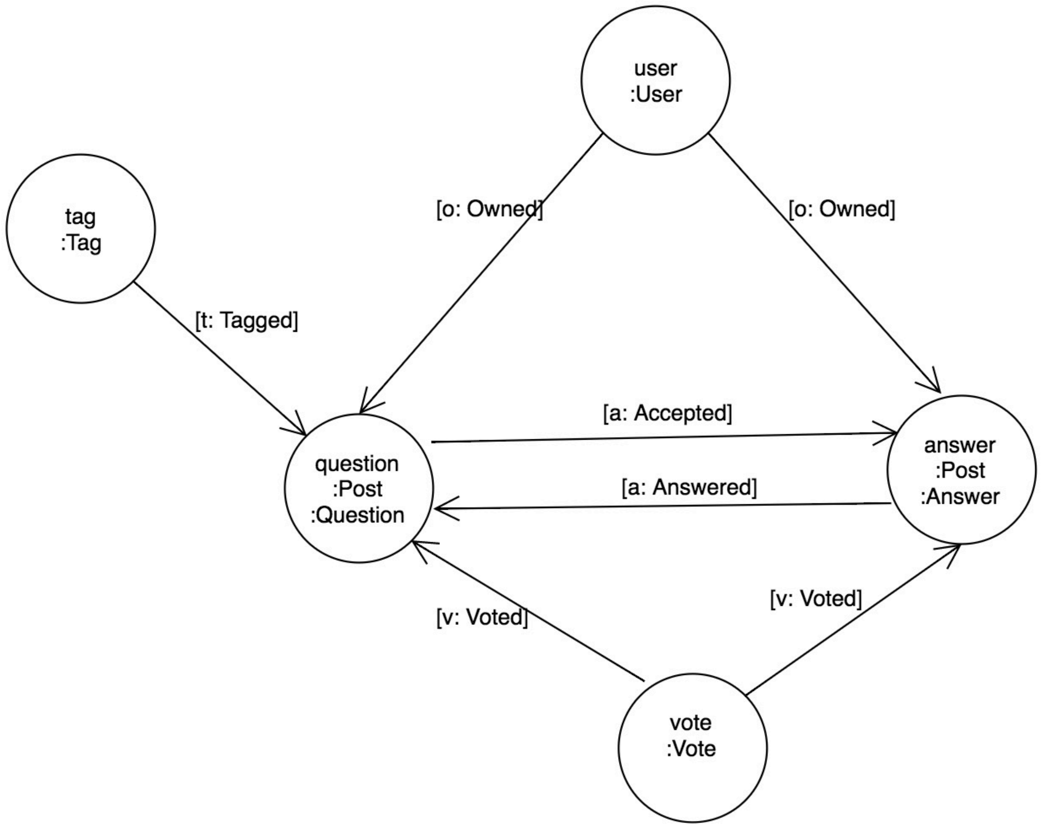


Figure 2.7 Relationships among four different entities in neo4j

For neo4j part, all the schemas for loading the data are in ***src/neo4j-helper.js*** file. We also convert the iso date into unix timestamp as neo4j does not support date by default. The user data attributes are also loaded into the User node in neo4j.

3 Query Design and Execution

3.1 Query Design

We are using aggregation for all the mongodb document query, take the third analytic query in the mongodb query, we first doing a $match to filter to get all the questions with accepted answer, then we do a $project filter to find out the questions has that tag, then we unwind the tags fields, then do a lookup to find out the accepted answers and their owners. At last we do the $group and $project to find that user accepted answers in a given topic.

```
[
  {
    $match: {
      PostTypeId: 1,
      AcceptedAnswerId: {"$exists": true, "$ne": null}
    }
  },
  {
    $project: {
      questionId: "$Id",
      questionTitle: "$Title",
      acceptedAnswerId: "$AcceptedAnswerId",
      tags: {
        $filter: {
          input: "$Tags",
          as: "item",
          cond: {
            "$setIsSubset": [["$$item.Id"], [4]]
          }
        }
      }
    }
  },
  {
    $match: {
      "tags.0": {"$exists": true}
    }
  },
  {
    $unwind: "$tags"
  },
  {
    $lookup: {
      from: "posts",
      localField: "acceptedAnswerId",
      foreignField: "Id",
      as: "acceptedAnswer"
    }
  },
  {
    $unwind: "$acceptedAnswer"
  },
  {
    $lookup: {
      from: "users",
      localField: "acceptedAnswer.OwnerUserId",
      foreignField: "Id",
         as: "acceptedUser"
      }
   },
   {
      $unwind: "$acceptedUser"
   },
   {
      $sort: {
         "acceptedUser.Id": 1
      }
   },
   {
      $group: {
         _id: {
            userId: "$acceptedUser.Id",
            displayName: "$acceptedUser.DisplayName"
         },
         acceptedQuestions: {
            $addToSet: {
               questionId: "$questionId",
               questionTitle: "$questionTitle",
               tags: "$tags"
            }
         }
      }
   },
   {
      $project: {
         _id: "$_id",
         acceptedQuestions: "$acceptedQuestions",
         numOfAcceptedQuestions: {
            $size: "$acceptedQuestions"
         }
      }
   },
   {
      $sort: {"numOfAcceptedQuestions": -1}
   },
   {
      $limit: 1
   }
]
```

While for the neo4j for this task, the query is quite simple, just need to find the owner of the accepted answers to the questions contains a tag

```
// Input Param, Please change it for different queries
const tagId = 4;

const query = `
   MATCH (user:User)-[:Owned]->(answer:Answer)<-[:Accepted]-(question:Question)-
[:Contains]->(tag:Tag{Id:${tagId}})
   WITH user, tag, count(*) as answersCount
   ORDER by answersCount DESC
   LIMIT 1
   MATCH (user)-[:Owned]->(answer:Answer)<-[:Accepted]-(question:Question)-[:Contains]->(tag)
   RETURN user.Id as userId, user.DisplayName as userDisplayName, question.Id as questionId, question.Title
as questionTitle
`;
```

3.2 Query Execution

There are two major execution steps:

- Loading data
- Executing Queries

First, you need to use Git to check this repo using the following command:

```
git clone https://github.com/MingxuanLi/comp5338-polygot-persistence-systems.git
```

After that you need to install the library dependencies we use to import and query data from MongoDB and neo4j, first you need to install **node.js 7.8.x** above and **npm 5.0** above here:
https://nodejs.org/en/download/current/
https://docs.npmjs.com/getting-started/installing-node
https://medium.com/@katopz/how-to-install-specific-nodejs-version-c6e1cec8aa11

Then run the following:

```
npm install
```

After the installing finish, you can try loading the data into MongoDB by running following:

```
npm run load -- --db=mongodb
```

And neo4j loading command is this:
npm run generate – (this does some conversion for the unix date format for neo4j)
npm run load -- --db=neo4j

After the loading is finished, then you can use the following query to fetch result.
For MongoDB: **npm run query -- --db=mongodb --query={$name}**
For neo4j: **npm run query -- --db=neo4j --query={$name}**

The variable $name can be **'sq1', 'sq2', 'aq1', 'aq2', 'aq3', 'aq4', 'aq5_1', 'aq5_2', 'aq6'**.
sq – simple query, **aq** – analytic query.

The result of the query should be printed in the terminal console.

Inside the query files in **src/mongo-queries**, you can change the input params in each file to query different result sets, neo4j queries are similar.

We are connnecting to localhost for mongodb and neo4j, there is NO ***user/pass*** for my local mongodb so you can find the url inside src/mongo-helper.js and I set ***neo4j/Password123*** as the neo4j login, you can find it under src/neo4j-helper.js

Section 4 Comparison and Summary

Overall speaking, the neo4j query is short and compact than the mongo query since it’s relational database so it’s quite easy to query data with relationships, also we observed that the performance of querying in neo4j is better than mongodb. Now we take the third analytic query for example.

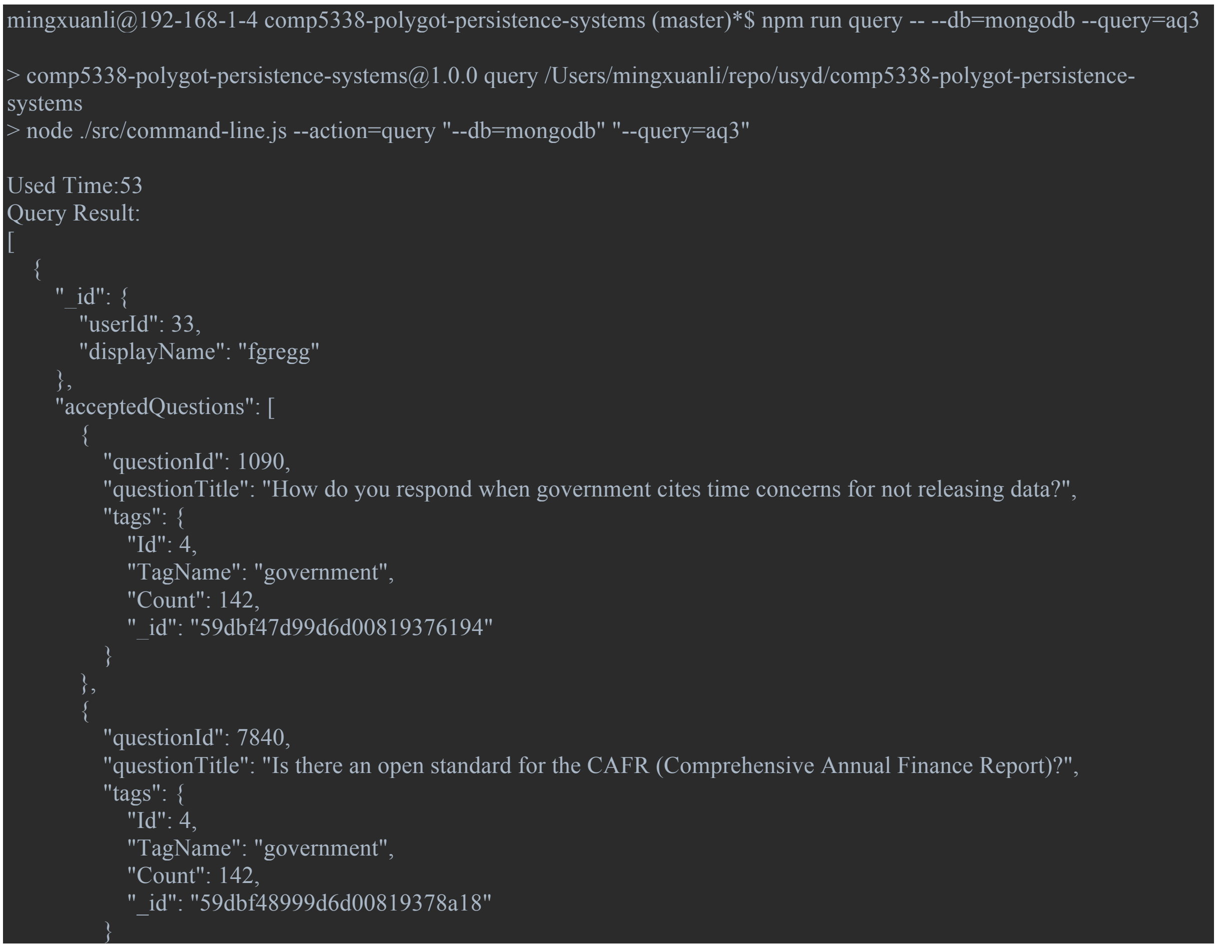
Secion 4.1 Query compare

The aggregation query on mongodb for querying the champion user questions (**analytic query 3**) is pretty long, we use a couple of $project with $filters, $unwind and $group to find the correct result set, those operations are pretty performance costly especially doing the $unwind, while the neo4j query for this question set is pretty simple and clean.

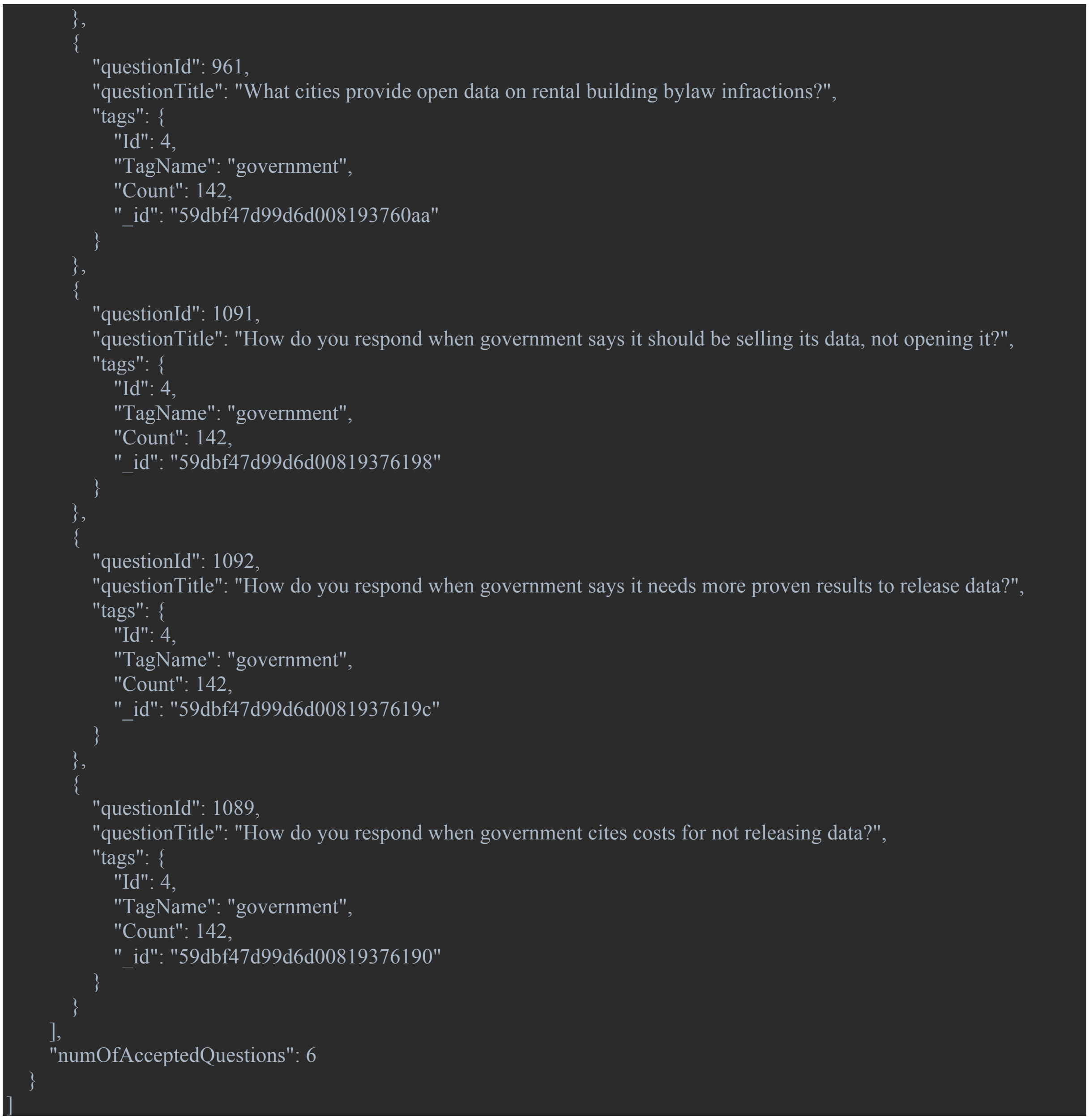
Section 4.2 Performance compare

Then we run both query in mongodb and neo4j by using our command line tool, see below, we can see the mongodb query cost ***53*** milliseconds to complete this query and return result set, but if we run the neo4j for same question set it only takes ***59*** milliseconds to fetch the champion user with their question list, we can see that neo4j is much faster than mongodb in handling relational graph database.

Running mongodb query result below:

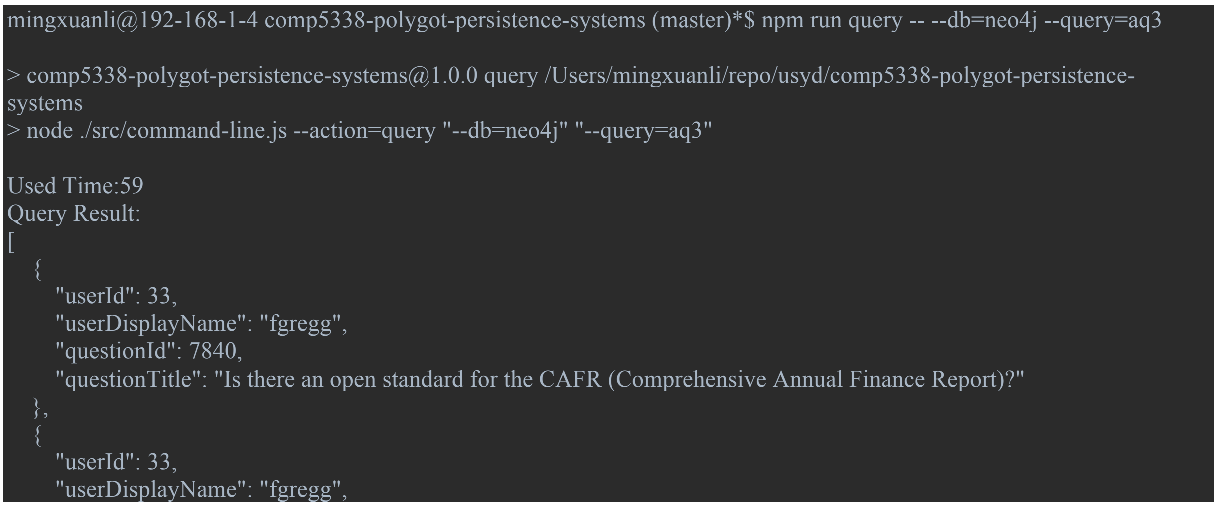


```
mingxuanli@192-168-1-4 comp5338-polygot-persistence-systems (master)*$ npm run query -- --db=mongodb --query=aq3

> comp5338-polygot-persistence-systems@1.0.0 query /Users/mingxuanli/repo/usyd/comp5338-polygot-persistence-
systems
> node ./src/command-line.js --action=query "--db=mongodb" "--query=aq3"

Used Time:53
Query Result:
[
   {
      "_id": {
         "userId": 33,
         "displayName": "fgregg"
      },
      "acceptedQuestions": [
         {
            "questionId": 1090,
            "questionTitle": "How do you respond when government cites time concerns for not releasing data?",
            "tags": {
               "Id": 4,
               "TagName": "government",
               "Count": 142,
               "_id": "59dbf47d99d6d00819376194"
            }
         },
         {
            "questionId": 7840,
            "questionTitle": "Is there an open standard for the CAFR (Comprehensive Annual Finance Report)?",
            "tags": {
               "Id": 4,
               "TagName": "government",
               "Count": 142,
               "_id": "59dbf48999d6d00819378a18"
            }
```



```
            },
            {
                "questionId": 961,
                "questionTitle": "What cities provide open data on rental building bylaw infractions?",
                "tags": {
                    "Id": 4,
                    "TagName": "government",
                    "Count": 142,
                    "_id": "59dbf47d99d6d008193760aa"
                }
            },
            {
                "questionId": 1091,
                "questionTitle": "How do you respond when government says it should be selling its data, not opening it?",
                "tags": {
                    "Id": 4,
                    "TagName": "government",
                    "Count": 142,
                    "_id": "59dbf47d99d6d00819376198"
                }
            },
            {
                "questionId": 1092,
                "questionTitle": "How do you respond when government says it needs more proven results to release data?",
                "tags": {
                    "Id": 4,
                    "TagName": "government",
                    "Count": 142,
                    "_id": "59dbf47d99d6d0081937619c"
                }
            },
            {
                "questionId": 1089,
                "questionTitle": "How do you respond when government cites costs for not releasing data?",
                "tags": {
                    "Id": 4,
                    "TagName": "government",
                    "Count": 142,
                    "_id": "59dbf47d99d6d00819376190"
                }
            }
        ],
        "numOfAcceptedQuestions": 6
    }
]
```

For neo4j, if we run the query for the third analytic query, the result is below:



```
mingxuanli@192-168-1-4 comp5338-polygot-persistence-systems (master)*$ npm run query -- --db=neo4j --query=aq3

> comp5338-polygot-persistence-systems@1.0.0 query /Users/mingxuanli/repo/usyd/comp5338-polygot-persistence-
systems
> node ./src/command-line.js --action=query "--db=neo4j" "--query=aq3"

Used Time:59
Query Result:
[
    {
        "userId": 33,
        "userDisplayName": "fgregg",
        "questionId": 7840,
        "questionTitle": "Is there an open standard for the CAFR (Comprehensive Annual Finance Report)?"
    },
    {
        "userId": 33,
        "userDisplayName": "fgregg",
        "questionId": 961,
        "questionTitle": "What cities provide open data on rental building bylaw infractions?"
    },
    {
        "userId": 33,
        "userDisplayName": "fgregg",
        "questionId": 1092,
        "questionTitle": "How do you respond when government says it needs more proven results to release data?"
    },
    {
        "userId": 33,
        "userDisplayName": "fgregg",
        "questionId": 1089,
        "questionTitle": "How do you respond when government cites costs for not releasing data?"
    },
    {
        "userId": 33,
        "userDisplayName": "fgregg",
        "questionId": 1090,
        "questionTitle": "How do you respond when government cites time concerns for not releasing data?"
    },
    {
        "userId": 33,
        "userDisplayName": "fgregg",
        "questionId": 1091,
        "questionTitle": "How do you respond when government says it should be selling its data, not opening it?"
    }
]
```

5 References

https://docs.mongodb.com/manual/applications/data-models/
https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1
https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-2
https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-3
https://neo4j.com/developer/javascript/
https://neo4j.com/developer/guide-data-modeling/