NoSQL Schema Design and Query Workload Implementation

Haoyue Li (470136555) Yang Ge (450005028)

October 2018

1 Introduction

In this assignment, we execute query tasks both on MongoDB and Neo4j and design suitable schema for these two system respectively. The implement of query and detail of design and the comparison of these two system will be illustrated as the following section.

2 MongoDB

2.1 Query Workload

We use MongoDB for query AQ2, AQ3, AQ4 in this experiment. After analysis, all three questions use the modified data structure is more suitable for solving the three questions.

For AQ2, we need to rank a tag by the number of users that has either posted a question in that tag or has answered a question in that tag. The modified data structure has value on the tag filed of the post, so we can easily count the number of user.

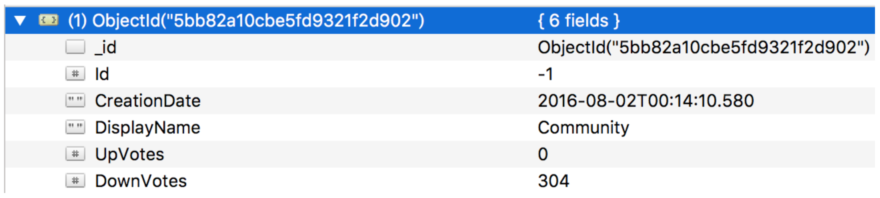
Both AQ3 ans AQ4 need to operate on the user whose answer post is accepted. So we create a new field to store the that user's id (AcceptedUser_Id), and these two queried will be simple to complete.

2.2 Schema Design

After analysis, we found that AQ2, AQ3 and AQ4 do not need to use the Tags and Votes collection. So we only need to import the Posts and Users collection into the MongoDB.

For Users collection, we find that some fields are unnecessary, so we remove the extra fields and only keep the useful fields for querying. Figure 1 shows the sample document of Users collection.

For Posts collection, we remove the unnecessary fields as well and add some important field that useful for the following querying. Figure 2 shows the sample document of Posts collection.



▼ (1) ObjectId("5bb82a10cbe5fd9321f2d902")	{ 6 fields }
_id	ObjectId("5bb82a10cbe5fd9321f2d902")
Id	-1
CreationDate	2016-08-02T00:14:10.580
DisplayName	Community
UpVotes	0
DownVotes	304

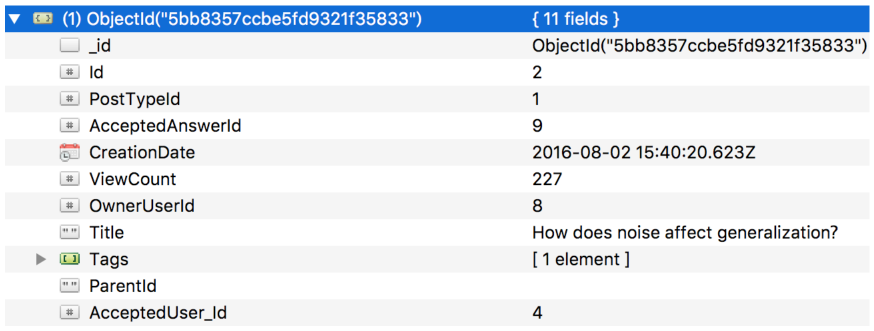
Figure 1: Users document in MongoDB

Firstly, since the original Posts collection do not have the information of tags in answer posts, we add the information of tags to each answer post. Each tags of answer post is consistent with it's corresponding question posts. So that, all posts include answer and question will have the information of tags.

And then, we create the field that is Id (AcceptedUser_Id) of the user whose answer was accepted on it's corresponding question post. If some questions do not accept any answer, these question posts will not have AcceptedUser_Id field. All answer posts also do not have this field.

Since the type of tags is string and the separator is a comma, we use $split function to split tags with a comma delimiter. The type of tags is converted from string to array.

Finally, we use $dateFromString function to convert the CreationDate field from String to date type. It is more effective to compare the date time on using date type.



▼ (1) ObjectId("5bb8357ccbe5fd9321f35833")	{ 11 fields }
_id	ObjectId("5bb8357ccbe5fd9321f35833")
Id	2
PostTypeId	1
AcceptedAnswerId	9
CreationDate	2016-08-02 15:40:20.623Z
ViewCount	227
OwnerUserId	8
Title	How does noise affect generalization?
▶ Tags	[1 element]
ParentId	
AcceptedUser_Id	4

Figure 2: Posts document in MongoDB

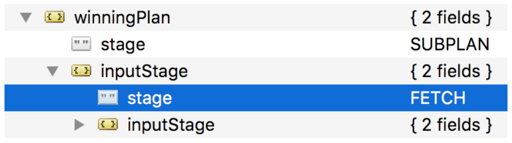
In order to improve the execution efficiency for these queries, we created some useful indexes for Posts collection. We set the index on Id, ParentId, Tags and CreationDate field, which will improve the speed of these queries and decrease the frequency of scan.

2.3 Query Design and Execution

2.3.1 SQ1

Using $lookup function to connect Posts collection and Users collection. The conditions for the connection is when the user id (OwnerUserId) of the Posts collection equal to the user id (Id) of the Users collection. Matching all documents for a given post id (Id and ParentId) and finding the relevant information for the owner user of these posts who either posted or answered the question.

When query a given user id, due to the index which is already created for Id field and ParentId field, it will match the documents according to index.



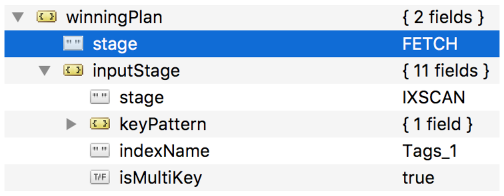
```
▼ winningPlan          { 2 fields }
    stage              SUBPLAN
  ▼ inputStage         { 2 fields }
      stage            FETCH
    ▶ inputStage       { 2 fields }
```

Figure 3: The winningPlan of SQ1

2.3.2 SQ2

Using $match function to find all posts have the given tag. Using $group function to group by post id and accumulating the view count. Sorting the sum of view counts in descending order and output the first one.

When query a given tag, due to the index which is already created for Tags field, it will match the documents according to index. Matching the specified documents according to the index instead of scanning the whole collection, which may improve efficiency and decrease the scan time.



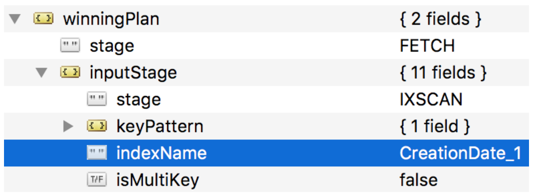
```
▼ winningPlan          { 2 fields }
    stage              FETCH
  ▼ inputStage         { 11 fields }
      stage            IXSCAN
    ▶ keyPattern       { 1 field }
      indexName        Tags_1
      isMultiKey       true
```

Figure 4: The winningPlan of SQ2

2.3.3 AQ2

Using $match function to match all posts which post in given time period. Since some posts have more than one tag, we need to using $unwind function to unwind tag field. So that, one tag will correspond to one post and we can group posts by tag. At the same time, we use $addToSet function to get the owner user id of these posts. That function can remove duplicate values, which avoid counting multiple posts posted by the same user. Using $size function to count the number of user in each tag. Finally, sorting the number of user and show the top 5.

When query a given start tine and end time, due to the index which is already created for CreationDate field, it will match the documents according to index. Matching the specified documents according to the index instead of scanning the whole collection, which may improve efficiency and decrease the scan time.



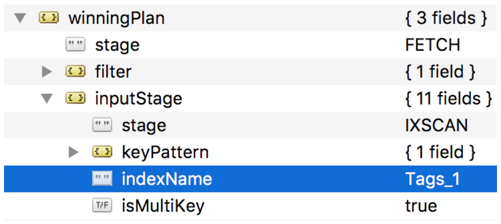
```
▼ winningPlan                 { 2 fields }
    stage                     FETCH
  ▼ inputStage                { 11 fields }
      stage                   IXSCAN
    ▶ keyPattern              { 1 field }
      indexName               CreationDate_1
      isMultiKey              false
```

Figure 5: The winningPlan of AQ2

2.3.4 AQ3

Matching all question posts that PostTypeId field equal to 1 have the given tag and the AnswerUser_id field exists. Grouping these posts by AnswerUser_id, and pushing post id and title into an array. After that, counting the number of posts that are accepted answers and sorting by it in descending order. Finally, selecting the top one.

When we want to find specified posts that have given tag, due to the index which is already created for tags field, it will match the documents according to index. Matching the specified documents according to the index instead of scanning the whole collection, which may improve efficiency and decrease the scan time.



```
▼ winningPlan                 { 3 fields }
    stage                     FETCH
  ▶ filter                    { 1 field }
  ▼ inputStage                { 11 fields }
      stage                   IXSCAN
    ▶ keyPattern              { 1 field }
      indexName               Tags_1
      isMultiKey              true
```

Figure 6: The winningPlan of AQ3

2.3.5 AQ4

After analysis, we find that we need to match tags which has the most accepted answer posted by given user and we need to store these tags. Therefore, we use python language to achieve that.

Firstly, entering the user id and matching all question posts that accepted answer is posted by the given user. Unwinding the tags field and grouping the posts by tag. And then, counting the numbber of posts and selecting the posts with a total number greater than a given values. After that, we will find the tags corresponding to these posts and store it to a list.

Next step, we use the list stored in the previous step to find the documents of the Posts collection that include those tags in that list. Selecting the posts that creation date less than the given date and sorting it by date in descending order.

This query need to find specified posts that have given tags and the creation date at given time. Since we create indexes on tags filed and CreationDate field, the query will match documents based on these index. Matching the specified documents according to the index instead of scanning the whole collection, which may improve efficiency and decrease the scan time.

3 Neo4j

3.1 Query Workload

We use Neo4j for query SQ1, SQ2, AQ1, AQ5, AQ6 in this experiment. The reason is that these queries are likely to be simple using relationship feature in Neo4j, especially for AQ6.

3.2 Schema Design

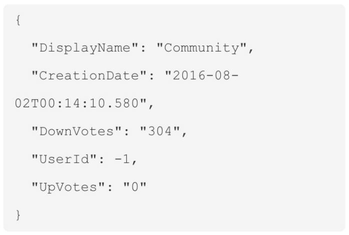
3.2.1 Pre-processing

We first generated corresponding csv files as we did not find a way to directly import the tsv files. We then deleted some attributes which are not used in our experiment, created some relationships and created corresponding indexes for each used node type based on the our use.

3.2.2 Node

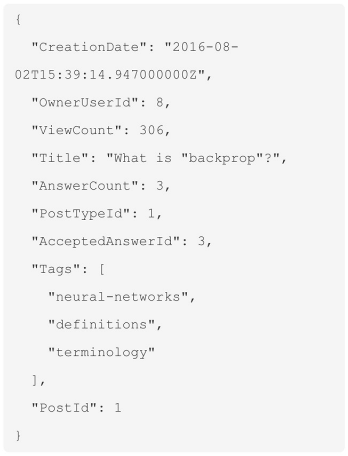
For User type of node, the attributes are $< id >$ which was automatically generated when importing the data, UserId, CreationDate, DisplayName, UpVotes, DownVotes. The primary key should be UserId, and we also set indexing on UserId.

For Post type of node, the attributes are $< id >$ which was automatically generated when importing the data, PostId, PostTypeId, AcceptedAnswerId, CreationDate, ViewCount, OwnerUserId, Title, Tags which was a array, AnswerCount, ParentId. The primary key should be PostId, and we also set indexing on PostId.



```
{
  "DisplayName": "Community",
  "CreationDate": "2016-08-
02T00:14:10.580",
  "DownVotes": "304",
  "UserId": -1,
  "UpVotes": "0"
}
```

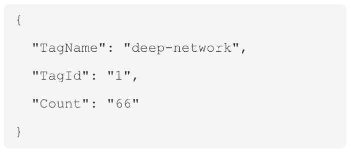
Figure 7: User document in Neo4j



```
{
  "CreationDate": "2016-08-
02T15:39:14.947000000Z",
  "OwnerUserId": 8,
  "ViewCount": 306,
  "Title": "What is "backprop"?",
  "AnswerCount": 3,
  "PostTypeId": 1,
  "AcceptedAnswerId": 3,
  "Tags": [
    "neural-networks",
    "definitions",
    "terminology"
  ],
  "PostId": 1
}
```

Figure 8: Post document in Neo4j

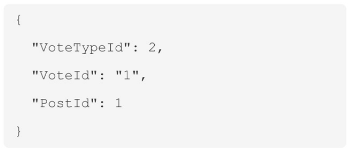
For Tag type of node, the attributes are $< id >$ which was automatically generated when importing the data, TagName, TagId, Count. Either TagName or TagId can be primary key, we set indexing on TagName in this experiment.



```
{
  "TagName": "deep-network",
  "TagId": "1",
  "Count": "66"
}
```

Figure 9: Tag document in Neo4j

For Vote type of node, the attributes are $< id >$ which was automatically generated when importing the data, VoteTypeId, VoteId, PostId. VoteId can be seen as primary key. In this experiment, we set VoteTypeId as index because we used this attribute in AQ5, did not use any others inside of Vote document.



```
{
  "VoteTypeId": 2,
  "VoteId": "1",
  "PostId": 1
}
```

Figure 10: Vote document in Neo4j

3.2.3 Relationship

In this experiment, we created 5 types of relationship.

The ”own” relationship means a user own a post.

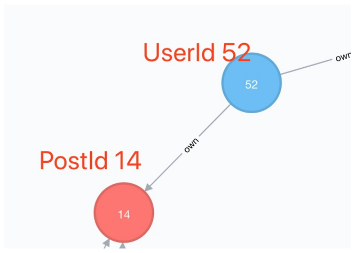


Figure 11: own relationship in Neo4j

The ”isparent” relationship means a post answered a question post.

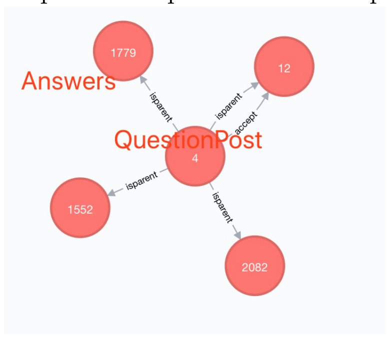


Figure 12: isparent relationship in Neo4j

The ”accept” relationship means a question post accept a answer post.

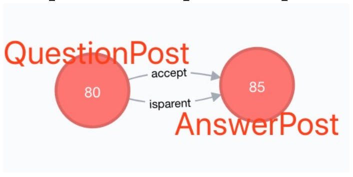


Figure 13: accept relationship in Neo4j

The ”haveTag” relationship means a question post have a particular tag.

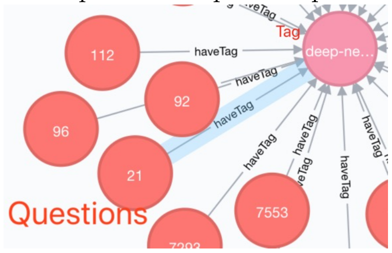


Figure 14: havetag relationship in Neo4j

The ”havevotes” relationship means a post have a particular vote.

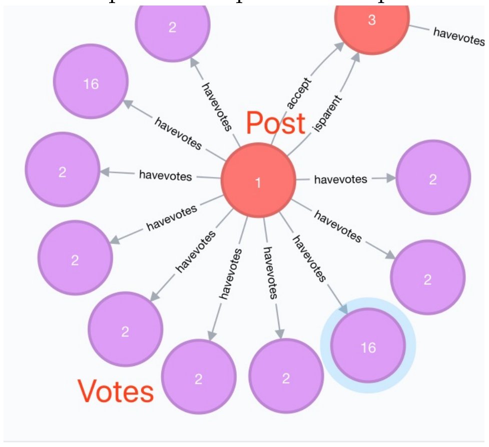


Figure 15: havevotes relationship in Neo4j

3.3 Query Design and Execution

3.3.1 SQ1

Given a question Id, the users involved in this question includes user posting that question and users answering that question.

In neo4j, the user posting this question can be matched by user $-[: own]->$ question post with question Id, the users answering that question can be matched by user $-[:\, own]- >$ a answer post $< -[:\, isparent]-$ the question post. After we combine these two matches, we would get $Match(u : User) - [r : own|isparent * 1..2] - (p : PostPostId : questionId)$.

In this query, the index PostId in Post can be used.

3.3.2 SQ2

We just try to find all the question post with a given tag, then sort them by ViewCount, and find the post with largest ViewCount.

In this query, the index TagName in tag can be used.

3.3.3 AQ1

Given a list of tags, we try to find the duration between all the questions within those tags and the accepted answers. Then we would get the tags and their minimum duration but no other information, so we match the relationship again and to find the questions within the tags which have the minimum duration and other information.

In this query, the index TagName in tag can be used.

3.3.4 AQ5

Given a specific α, we first find all questions whose upvote count are greater than that α, then we try to find the maximum upvote count in the answers for each question, then we find the accepted answer upvote count, compare the maxium upvote count and accepted answer upvote count and return.

In this query, the indexes VoteTypeId in vote, PostId in post, can be used.

3.3.5 AQ6

Say u2 is a coauthor of a specific user u1, there would be three cases: u1 post a question, u2 answer it; u2 post a question, u1 answer it; u1 and u2 both answer a same question. In this experiment, this relationship could be matched by $Match(u : User) - [r1 : own]- > (p1 : Post) - [r2 : isparent * 1..2] - (p2 : Post) < -[r3 : own] - (u2 : User)$. After matching, we just set a specific user and find all coauthors and find the top coauthors.

In this query, the indexe UserId in user can be used.

4 Comparison and Summary

They are both NoSQL database, but MongoDB is a graph database while Neo4j is a graph database. Does your data fit the document model? Then MongoDB is the better choice. Does it fit the graph database model? Then Neo4j is the better choice. If data fit the document model, then MongoDb would be the better choice, if data fit the graph database model, that means the data have strong relationship property in data or need to do path traversal or shortest path computations.

4.1 ease of use

For data with strong relationship property, Neo4j may be easier in use, because some Cypher clauses are very similar to SQL clauses, that may let user quickly understand it. The relationship property in Neo4j is also easy to understand, and Neo4j performance very well in strong relationship task. For MongoDB, we need to use "*lookup*" to connect two collections, which is harder to use and to understand.

For data with document property, Neo4j and MongoDB are both easy to use. However, in that case, MongoDB would be a better choice since it is designed for document data, and the advantage of Neo4j, i.e. relationship, is not very useful.

4.2 query performance

For single collection query in MongoDB and single node type query in Neo4j, MongoDB has better performance. For query need to use two or more collection in MongoDB and need to use relationship in Neo4j, Neo4j has better performance.

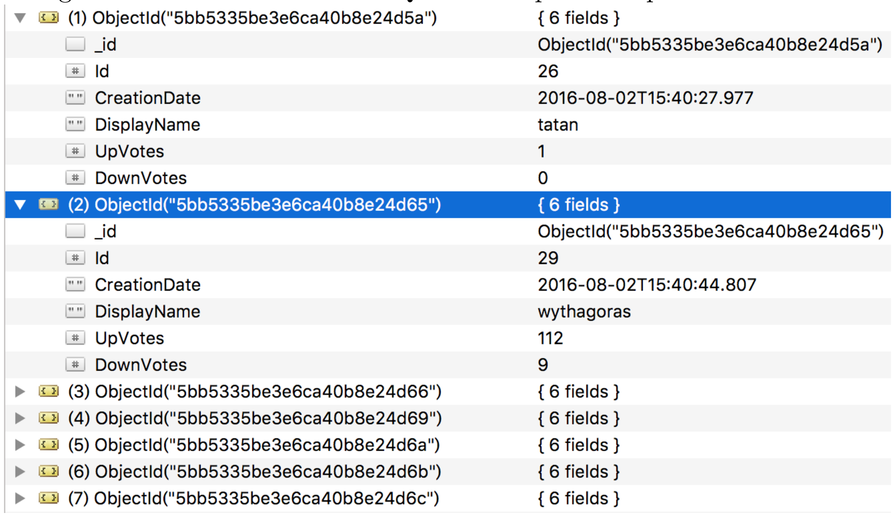
4.3 schema differences

The key difference is the relationship in Neo4j, as MongoDB does not create relationships between the database models, as each data set stored in the document store of the database is independent. A graph system like Neo4j requires handling the complex relationship of the database.

5 Sample results

5.1 MongoDB

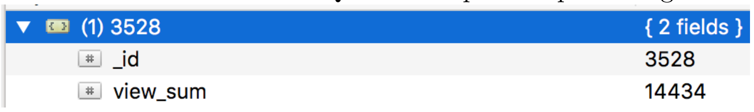
The figure 16 shows the result of SQ1 which input 7 for post id.



```
▼ (1) ObjectId("5bb5335be3e6ca40b8e24d5a")    { 6 fields }
    _id                                       ObjectId("5bb5335be3e6ca40b8e24d5a")
    Id                                        26
    CreationDate                              2016-08-02T15:40:27.977
    DisplayName                               tatan
    UpVotes                                   1
    DownVotes                                 0
▼ (2) ObjectId("5bb5335be3e6ca40b8e24d65")    { 6 fields }
    _id                                       ObjectId("5bb5335be3e6ca40b8e24d65")
    Id                                        29
    CreationDate                              2016-08-02T15:40:44.807
    DisplayName                               wythagoras
    UpVotes                                   112
    DownVotes                                 9
▶ (3) ObjectId("5bb5335be3e6ca40b8e24d66")    { 6 fields }
▶ (4) ObjectId("5bb5335be3e6ca40b8e24d69")    { 6 fields }
▶ (5) ObjectId("5bb5335be3e6ca40b8e24d6a")    { 6 fields }
▶ (6) ObjectId("5bb5335be3e6ca40b8e24d6b")    { 6 fields }
▶ (7) ObjectId("5bb5335be3e6ca40b8e24d6c")    { 6 fields }
```

Figure 16: The result of SQ1

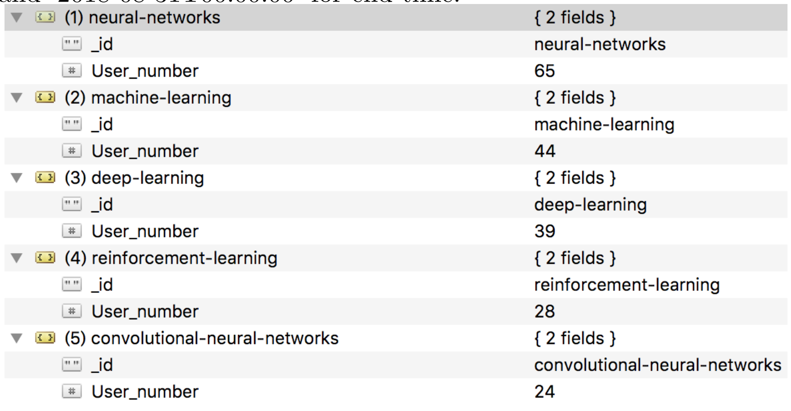
The figure 17 shows the result of SQ2 which input 'deep-learning' for tag.



```
▼ (1) 3528          { 2 fields }
    _id             3528
    view_sum        14434
```

Figure 17: The result of SQ2

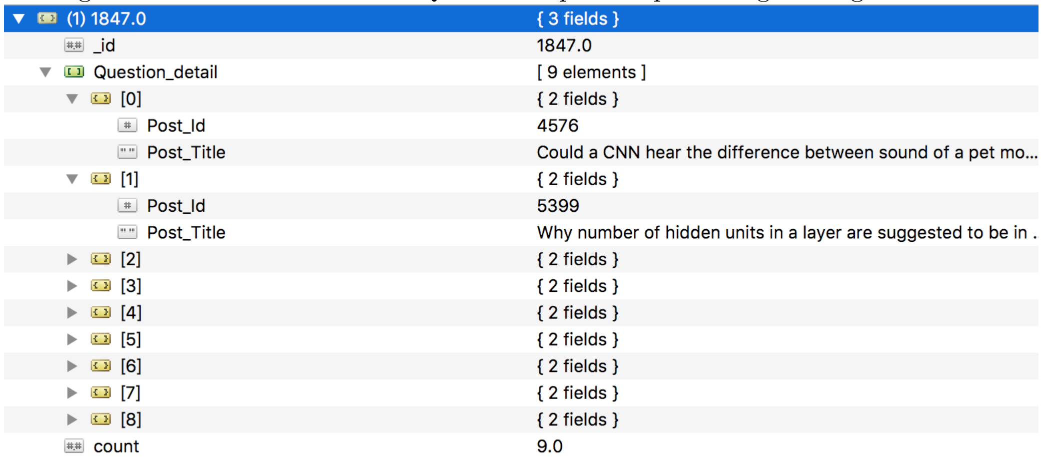
Figure 18 shows the result of AQ2 which input '2018-08-01T00:00:00' for start time and '2018-08-31T00:00:00' for end time.



```
▼ (1) neural-networks                  { 2 fields }
    _id                                neural-networks
    User_number                        65
▼ (2) machine-learning                 { 2 fields }
    _id                                machine-learning
    User_number                        44
▼ (3) deep-learning                    { 2 fields }
    _id                                deep-learning
    User_number                        39
▼ (4) reinforcement-learning           { 2 fields }
    _id                                reinforcement-learning
    User_number                        28
▼ (5) convolutional-neural-networks    { 2 fields }
    _id                                convolutional-neural-networks
    User_number                        24
```

Figure 18: The result of AQ2

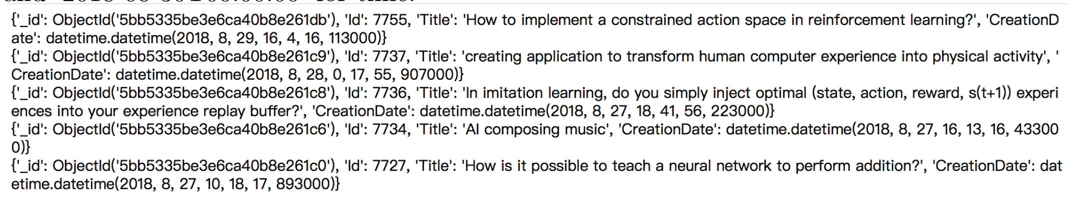
Figure 19 shows the result of AQ3 which input 'deep-learning' for tag.



```
▼ (1) 1847.0                          { 3 fields }
    _id                               1847.0
  ▼ Question_detail                   [ 9 elements ]
    ▼ [0]                             { 2 fields }
        Post_Id                       4576
        Post_Title                    Could a CNN hear the difference between sound of a pet mo...
    ▼ [1]                             { 2 fields }
        Post_Id                       5399
        Post_Title                    Why number of hidden units in a layer are suggested to be in .
    ▶ [2]                             { 2 fields }
    ▶ [3]                             { 2 fields }
    ▶ [4]                             { 2 fields }
    ▶ [5]                             { 2 fields }
    ▶ [6]                             { 2 fields }
    ▶ [7]                             { 2 fields }
    ▶ [8]                             { 2 fields }
    count                             9.0
```

Figure 19: The result of AQ3

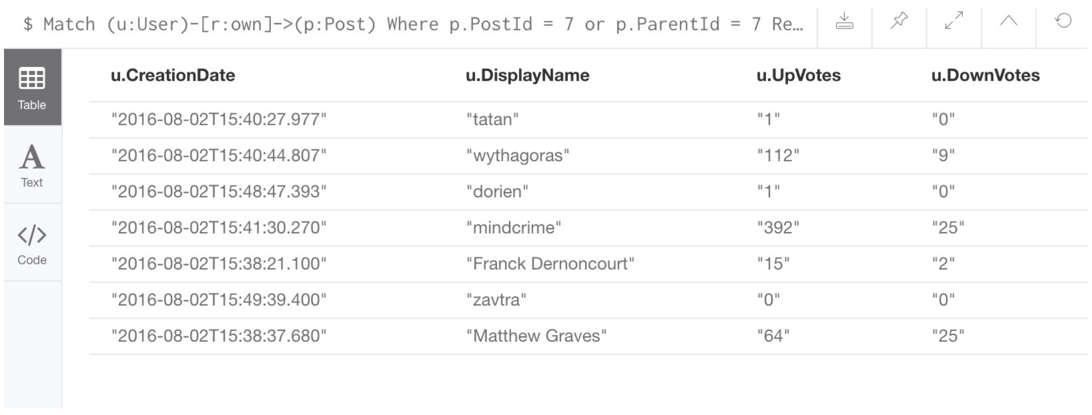
Figure 20 shows the result of AQ4 which input 4398 for user id, 4 for threshold and '2018-08-30T00:00:00' for time.



```
{'_id': ObjectId('5bb5335be3e6ca40b8e261db'), 'Id': 7755, 'Title': 'How to implement a constrained action space in reinforcement learning?', 'CreationD
ate': datetime.datetime(2018, 8, 29, 16, 4, 16, 113000)}
{'_id': ObjectId('5bb5335be3e6ca40b8e261c9'), 'Id': 7737, 'Title': 'creating application to transform human computer experience into physical activity', '
CreationDate': datetime.datetime(2018, 8, 28, 0, 17, 55, 907000)}
{'_id': ObjectId('5bb5335be3e6ca40b8e261c8'), 'Id': 7736, 'Title': 'In imitation learning, do you simply inject optimal (state, action, reward, s(t+1)) experi
ences into your experience replay buffer?', 'CreationDate': datetime.datetime(2018, 8, 27, 18, 41, 56, 223000)}
{'_id': ObjectId('5bb5335be3e6ca40b8e261c6'), 'Id': 7734, 'Title': 'AI composing music', 'CreationDate': datetime.datetime(2018, 8, 27, 16, 13, 16, 43300
0)}
{'_id': ObjectId('5bb5335be3e6ca40b8e261c0'), 'Id': 7727, 'Title': 'How is it possible to teach a neural network to perform addition?', 'CreationDate': dat
etime.datetime(2018, 8, 27, 10, 18, 17, 893000)}
```

Figure 20: The result of AQ4

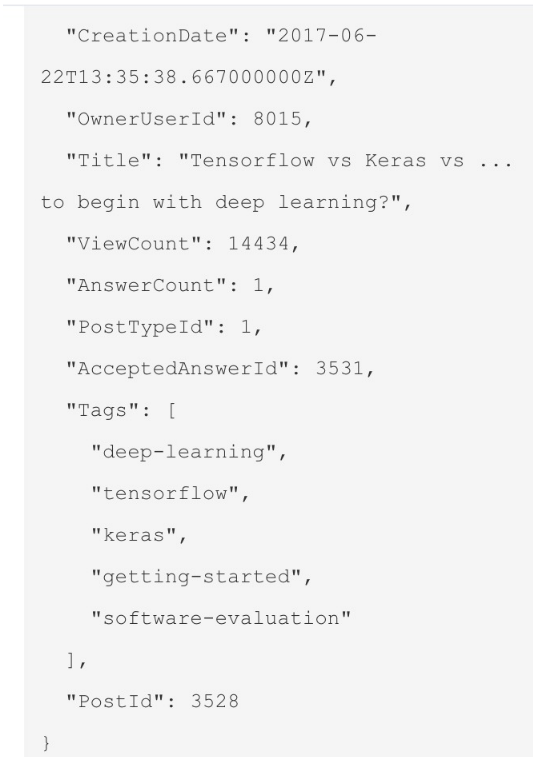
5.2 Neo4j



```
$ Match (u:User)-[r:own]->(p:Post) Where p.PostId = 7 or p.ParentId = 7 Re…
```

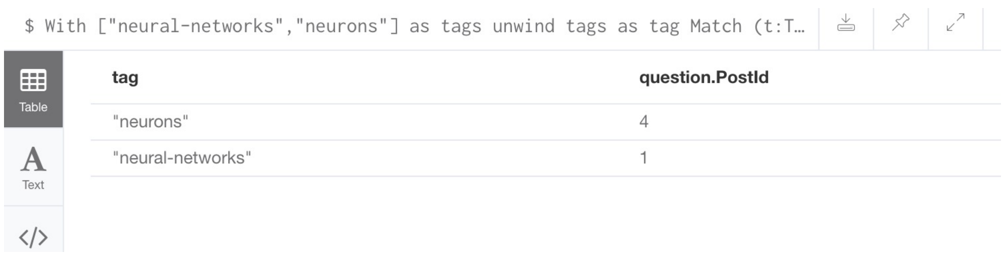
u.CreationDate	u.DisplayName	u.UpVotes	u.DownVotes
"2016-08-02T15:40:27.977"	"tatan"	"1"	"0"
"2016-08-02T15:40:44.807"	"wythagoras"	"112"	"9"
"2016-08-02T15:48:47.393"	"dorien"	"1"	"0"
"2016-08-02T15:41:30.270"	"mindcrime"	"392"	"25"
"2016-08-02T15:38:21.100"	"Franck Dernoncourt"	"15"	"2"
"2016-08-02T15:49:39.400"	"zavtra"	"0"	"0"
"2016-08-02T15:38:37.680"	"Matthew Graves"	"64"	"25"

Figure 21: SQ1 by Neo4j Question Id 7



```
  "CreationDate": "2017-06-
22T13:35:38.667000000Z",
  "OwnerUserId": 8015,
  "Title": "Tensorflow vs Keras vs ...
to begin with deep learning?",
  "ViewCount": 14434,
  "AnswerCount": 1,
  "PostTypeId": 1,
  "AcceptedAnswerId": 3531,
  "Tags": [
    "deep-learning",
    "tensorflow",
    "keras",
    "getting-started",
    "software-evaluation"
  ],
  "PostId": 3528
}
```

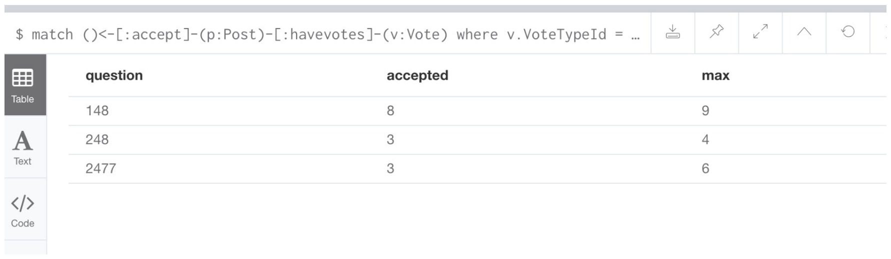
Figure 22: SQ2 by Neo4j Tag deep learning



```
$ With ["neural-networks","neurons"] as tags unwind tags as tag Match (t:T…
```

tag	question.PostId
"neurons"	4
"neural-networks"	1

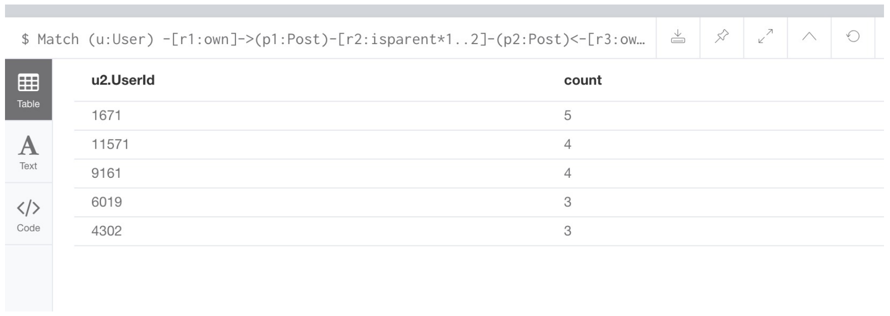
Figure 23: AQ1 by Neo4j Tags neural-networks and neurons



```
$ match ()<-[:accept]-(p:Post)-[:havevotes]-(v:Vote) where v.VoteTypeId = …
```

question	accepted	max
148	8	9
248	3	4
2477	3	6

Figure 24: AQ5 by Neo4j alpha 10



```
$ Match (u:User) -[r1:own]->(p1:Post)-[r2:isparent*1..2]-(p2:Post)<-[r3:ow…
```

u2.UserId	count
1671	5
11571	4
9161	4
6019	3
4302	3

Figure 25: AQ6 by Neo4j User Id 4398