CM3104: Large-Scale Databases Report

21050251

Theodor Baur

Content

Question 1: MongoDB

1: Designing an embedded data model

Summary

The *database_B* model contains the 5 original collections (*answers*, *badges*, *comments*, *questions*, and *users*). The collections contain an embedded document inside if the original collection from *database_A* contains an ID field referring to another document present in the dataset. IDs are preserved so that we are able to see cases where the related document exists but isn't in the dataset, these fields can be deleted if we had access to the complete dataset.

Answers

Sample


```
_id: ObjectId('614ca7926378ff93c07f43b6')
id: 53964041
body: "<p>Here's how I've implemented it now:</p>

      <ul>
      <li>Create a Subclass…"
comment_count: 0
creation_date: 2018-12-28T20:38:33.670+00:00
last_activity_date: 2018-12-28T20:47:03.833+00:00
last_edit_date: 2018-12-28T20:47:03.833+00:00
last_editor_user_id: 91
owner_user_id: 91
parent_id: 53960223
post_type_id: 2
score: 1
▸ parent_question: Object
▸ last_editor: Object
▸ owner: Object
```

Query

```
dbA.answers.find().forEach(function(answer) {
    var relatedQuestion = dbA.questions.findOne({
        id: answer.parent_id
    });

    var relatedLastEditor = dbA.users.findOne({
        id: answer.last_editor_user_id
    });

    var relatedOwner = dbA.users.findOne({
        id: answer.owner_user_id
    });

    answer.parent_question = relatedQuestion;
    answer.last_editor = relatedLastEditor;
    answer.owner = relatedOwner;

    dbB.answers.insert(answer);
});
```

Description

The original *answers* collection fields are preserved from *database_A* with the addition of the embedded documents: *parent_question, last_editor,* and *owner*. When creating *database_B*, the field value *last_editor_user_id* is checked against the *users* collection from *database_A*. If there is a *user* present in the dataset then the *user* document is embedded into the field *last_editor*, otherwise the *last_editor* field is set to *null*. The same process is applied when creating the *parent_question* (checked against the *questions* collection), and *owner* (checked against the *users* collection) fields. The *last_editor_user_id, owner_user_id,* and *parent_id* fields are preserved.

Badges

Sample


```
_id: ObjectId('614ca7966378ff93c07fd7e1')
id: 1153467
name: "Nice Question"
date: 2010-10-25T18:19:04.783+00:00
user_id: 91
class: 3
tag_based: false
▸ user: Object
```


Query

```
dbA.badges.find().forEach(function(badge) {
    var relatedUser = dbA.users.findOne({
        id: badge.user_id
    });

    badge.user = relatedUser;

    dbB.badges.insert(badge);
});
```

Description

All of the original *badges* collection fields are preserved from *database_A* with the addition of the embedded document *user*. The *user* field is created by checking the existing *user_id* field against *database_A's users* collection. The *user_id* field itself is preserved.

Comments

Sample


```
_id: ObjectId('614ca7946378ff93c07f80dc')
id: 84230034
text: "@MickyD In this case, it's not for HTTPS cert but to prototype and tes…"
creation_date: 2018-02-05T06:00:37.640+00:00
post_id: 48615537
user_id: 91
score: 0
▸ user: Object
▸ post: Object
```

Query

```
dbA.comments.find().forEach(function(comment) {
    var relatedUser = dbA.users.findOne({
        id: comment.user_id
    });

    var relatedPost = null;

    var relatedPost = dbA.questions.findOne({
        id: comment.post_id
    });

    if (!relatedPost) {
        relatedPost = dbA.answers.findOne({
            id: comment.post_id
        });
    }

    comment.user = relatedUser;
    comment.post = relatedPost;

    dbB.comments.insert(comment);
});
```

Description

All of the original *comments* collection fields are preserved from *database_A* with the addition of the the two embedded documents: *user* and *post*. The *user* field is created by checking the existing *user_id* field against *database_A's users* collection. The *post* field is created by checking the existing *post_id* against the *id* of the *questions* collection from *database_A*, then if no existing *questions* are found for that *id* it is checked against the *answers* collection of *database_A*, if no *answers* are found relating to the *id* then the field is kept as a null value. The original *user_id* and *post_id* fields are preserved.

Questions

Sample


```
_id: ObjectId('614ca7916378ff93c07f357a')
id: 4610733
title: "Script-Based Configuration in .net?"
body: "<p>One of the downsides of web.config/app.config is that it's just Mag…"
accepted_answer_id: 5087724
answer_count: 2
comment_count: 5
creation_date: 2011-01-06T00:36:52.593+00:00
favorite_count: 1
last_activity_date: 2011-02-23T06:27:41.367+00:00
last_edit_date: 2011-01-06T01:43:49.503+00:00
last_editor_user_id: 91
owner_user_id: 91
post_type_id: 1
score: 2
tags: ".net|web-config"
view_count: 283
▸ accepted_answer: Object
▸ last_editor: Object
▸ owner: Object
```


Query

```
dbA.questions.find().forEach(function(question) {
    var acceptedAnswer = dbA.answers.findOne({
      id: question.accepted_answer_id
    });

    var relatedLastEditor = dbA.users.findOne({
        id: question.last_editor_user_id
    });

    var relatedOwner = dbA.users.findOne({
          id: question.owner_user_id
    });

    question.accepted_answer = acceptedAnswer;
    question.last_editor = relatedLastEditor;
    question.owner = relatedOwner;

    dbB.questions.insert(question);
});
```

Description

All of the original *questions* collection fields are preserved from *database_A* with the addition of the embedded documents: *accepted_answer*, *last_editor*, and *owner*. The field value *accepted_answer_id* is checked against the *answers*

field from *database_A*. If there is an *answer* present in the dataset relating to the *accepted_answer_id* it is embedded into the *accepted_answer* field, otherwise it is set to *null*. The same process is applied when creating the *last_editor* and *owner* (both checked against the *users* collection) fields. The *last_editor_user_id, owner_user_id,* and *accepted_answer_id* fields are all preserved.

Users

Sample

```
_id: ObjectId('614ca7906378ff93c07f345d')
id: 91
display_name: "Michael Stum"
about_me: "<p>The same thing we do every night, Pinky. Try to take over the world…"
age: null
creation_date: 2008-08-01T17:55:22.200+00:00
last_access_date: 2021-09-03T16:04:32.200+00:00
location: "Raleigh, NC, United States"
reputation: 169147
up_votes: 7245
down_votes: 55
views: 13852
profile_image_url: null
website_url: "https://www.Stum.de"
```

Query

```
dbA.users.find().forEach(function(user) {
    dbB.users.insert(user);
});
```

Description

The *users* collection is kept identical to *database_A* as keeping documents embedded is likely to result in each user document containing many fields with multiple embedded documents, which can be seen as an inefficient use of space and may affect query performance. For example, an *answers* field inside the *user* collection could contain hundreds of individual *answers*, the same goes for a *badges, comments,* or *questions* field.

2: Writing Queries

i)

Database	database_A	database_B
Query	dbA.badges.aggregate([{ $match: { name: "Nice Question", date: { $gte: ISODate("2020-01-01"), }, }, }, { $lookup: { from: "users", localField: "user_id", foreignField: "id", as: "user", }, }, { $unwind: "$user", }, { $lookup: { from: "questions", localField: "user.id", foreignField: "owner_user_id", as: "questions", }, }, { $unwind: "$questions", }, { $match: { "questions.accepted_answer_id": { $nin: [NaN, null], }, }, }, { $lookup: { from: "answers", localField: "questions.accepted_answer_id", foreignField: "id", as: "accepted_answer", }, }, { $unwind: { path: "$accepted_answer", preserveNullAndEmptyArrays: true,	dbB.badges.aggregate([{ $match: { name: "Nice Question", date: { $gte: ISODate("2020-01-01"), }, }, }, { $lookup: { from: "questions", localField: "user.id", foreignField: "owner_user_id", as: "questions", }, }, { $unwind: "$questions", }, { $match: { "questions.accepted_answer_id": { $nin: [NaN, null], }, }, }, { $project: { _id : 0, user_id: "$user.id", user_display_name: "$user.display_name", badge_name: "$name", badge_date: "$date", question_id: "$questions.id", question_title: "$questions.title", accepted_answer_id: "$questions.accepted_answer_id", answer_score:{$ifNull: ["$questions.accepted_answer.score", null]}, }, },])

	`` }, }, { $project: { _id : 0, user_id: "$user.id", user_display_name: "$user.display_name", badge_name: "$name", badge_date: "$date", question_id: "$questions.id", question_title: "$questions.title", accepted_answer_id: "$questions.accepted_answer_id", answer_score:{$ifNull: ["$accepted_answer.score", null]}, }, },]) ```	
Number of results	1142	1142
Sample data set (limit 10 records)	``` { "user_id": 91, "user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 9440546, "question_title": "Does Java support something like Type Forwarding?", "accepted_answer_id": 9440676, "answer_score": null }, { "user_id": 91, "user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 51345161, "question_title": "Should I take ILogger, ILogger<T>, ILoggerFactory or ILoggerProvider for a library?", "accepted_answer_id": 51394689, "answer_score": null }, { "user_id": 91, "user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 21058775, "question_title": "Can I get the current seed from a Mersenne Twister?", "accepted_answer_id": 21058836, ```	``` { user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08- 03T20:41:57.170Z"), question_id: 9440546, question_title: 'Does Java support something like Type Forwarding?', accepted_answer_id: 9440676, answer_score: null }, { user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08- 03T20:41:57.170Z"), question_id: 51345161, question_title: 'Should I take ILogger, ILogger<T>, ILoggerFactory or ILoggerProvider for a library?', accepted_answer_id: 51394689, answer_score: null }, { user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08- 03T20:41:57.170Z"), question_id: 21058775, question_title: 'Can I get the current seed from a Mersenne Twister?', accepted_answer_id: 21058836, answer_score: null }, { ```

	"answer_score": null }, { "user_id": 91, "user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 58675200, "question_title": "In GitHub Actions, can I return back a value to be used as a condition later?", "accepted_answer_id": 58676568, "answer_score": null }, { "user_id": 91, "user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 32955658, "question_title": "What is the correct way to use wide-char ncurses on Debian/Ubuntu?", "accepted_answer_id": 32959643, "answer_score": null }, { "user_id": 91, "user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 24583204, "question_title": "What is the Client/Server model when using Electron (Atom Shell)?", "accepted_answer_id": 24618996, "answer_score": null }, { "user_id": 91, "user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 31083378, "question_title": "Is there a standard algorithm to balance overlapping objects into buckets?", "accepted_answer_id": 31084061, "answer_score": null }, { "user_id": 91,	user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08-03T20:41:57.170Z"), question_id: 58675200, question_title: 'In GitHub Actions, can I return back a value to be used as a condition later?', accepted_answer_id: 58676568, answer_score: null }, { user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08-03T20:41:57.170Z"), question_id: 32955658, question_title: 'What is the correct way to use wide-char ncurses on Debian/Ubuntu?', accepted_answer_id: 32959643, answer_score: null }, { user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08-03T20:41:57.170Z"), question_id: 24583204, question_title: 'What is the Client/Server model when using Electron (Atom Shell)?', accepted_answer_id: 24618996, answer_score: null }, { user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08-03T20:41:57.170Z"), question_id: 31083378, question_title: 'Is there a standard algorithm to balance overlapping objects into buckets?', accepted_answer_id: 31084061, answer_score: null }, { user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08-03T20:41:57.170Z"), question_id: 49263968,

	<pre>"user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 49263968, "question_title": "Does \"fixed\" get cleaned up properly if an Exception is thrown?", "accepted_answer_id": 49270251, "answer_score": null }, { "user_id": 91, "user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 18008224, "question_title": "What's the recommended XCode to build PPC/Tiger binaries?", "accepted_answer_id": 18008256, "answer_score": null }, { "user_id": 91, "user_display_name": "Michael Stum", "badge_name": "Nice Question", "badge_date": { "$date": "2020-08-03T20:41:57.170Z" }, "question_id": 63077017, "question_title": "How should I pass an array of strings to a C library using P/Invoke?", "accepted_answer_id": 63079098, "answer_score": null }</pre>	<pre>question_title: 'Does "fixed" get cleaned up properly if an Exception is thrown?', accepted_answer_id: 49270251, answer_score: null }, { user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08-03T20:41:57.170Z"), question_id: 18008224, question_title: "What's the recommended XCode to build PPC/Tiger binaries?", accepted_answer_id: 18008256, answer_score: null }, { user_id: 91, user_display_name: 'Michael Stum', badge_name: 'Nice Question', badge_date: ISODate("2020-08-03T20:41:57.170Z"), question_id: 63077017, question_title: 'How should I pass an array of strings to a C library using P/Invoke?', accepted_answer_id: 63079098, answer_score: null }</pre>

ii)

Database	database_A	database_B
Query	<pre>dbA.badges.aggregate([{ $match: { name: "Nice Question", date: { $gte: ISODate("2020-01-01"), }, }, }, { $lookup: { from: "users", localField: "user_id", foreignField: "id", as: "user", },</pre>	<pre>dbB.badges.aggregate [{ $match: { name: "Nice Question", date: { $gte: ISODate("2020-01-01"), }, }, }, { $lookup: { from: "questions", localField: "user.id", foreignField: "owner_user_id", as: "questions", },</pre>

```
  },
  {
    $unwind: "$user",
  },
  {
    $lookup: {
      from: "questions",
      localField: "user_id",
      foreignField: "owner_user_id",
      as: "questions",
    },
  },
  {
    $unwind: "$questions",
  },
  {
    $group: {
      _id: "$questions.accepted_answer_id",
      accepted_answer_id: {
        $first:
"$questions.accepted_answer_id",
      },
      questions_id: {
        $first: "$questions.id",
      },
      user_id: {
        $first: "$user.id",
      },
    },
  },
  {
    $match: {
      accepted_answer_id: {
        $nin: [NaN, null],
      },
    },
  },
  {
    $lookup: {
      from: "answers",
      localField: "accepted_answer_id",
      foreignField: "id",
      as: "accepted_answer",
    },
  },
  {
    $unwind: {
      path: "$accepted_answer",
      preserveNullAndEmptyArrays: true,
    },
  },
  {
    $lookup: {
      from: "comments",
      localField: "accepted_answer_id",
      foreignField: "post_id",
      as: "accepted_answer_comments",
    },
  },
  {
```

```
  },
  {
    $unwind: "$questions",
  },
  {
    $group: {
      _id: "$questions.accepted_answer_id",
      accepted_answer_id: {
        $first:
"$questions.accepted_answer_id",
      },
      question_id: {
        $first: "$questions.id",
      },
      user_id: {
        $first: "$user.id",
      },
    },
  },
  {
    $match: {
      accepted_answer_id: {
        $nin: [NaN, null],
      },
    },
  },
  {
    $lookup: {
      from: "comments",
      localField: "accepted_answer_id",
      foreignField: "post_id",
      as: "accepted_answer_comments",
    },
  },
  {
    $unwind: "$accepted_answer_comments",
  },
  {
    $project: {
      _id: 0,
      user_id: "$user_id",
      question_id: "$question_id",
      accepted_answer_id: {$ifNull:
["$accepted_answer_id", null]},
      comment_id: {$ifNull:
["$accepted_answer_comments.id", null]},
    },
  },
])
```

	$unwind: "$accepted_answer_comments", }, { $project: { _id: 0, user_id: "$user_id", question_id: "$questions_id", accepted_answer_id: {$ifNull: ["$accepted_answer_id", null]}, comment_id: {$ifNull: ["$accepted_answer_comments.id", null]}, }, },])	
Number of results	375	375
Sample data set (limit 10 records)	{ user_id: 91, question_id: 1520933, accepted_answer_id: 1529352, comment_id: 1387080 }, { user_id: 32495, question_id: 30007523, accepted_answer_id: 30008979, comment_id: 48158695 }, { user_id: 99694, question_id: 2394944, accepted_answer_id: 2394953, comment_id: 2374157 }, { user_id: 91, question_id: 63077017, accepted_answer_id: 63079098, comment_id: 111548995 }, { user_id: 1912, question_id: 9380069, accepted_answer_id: 9380702, comment_id: 11849183 }, { user_id: 1912, question_id: 9380069, accepted_answer_id: 9380702, comment_id: 11849222 }, { user_id: 91, question_id: 352674, accepted_answer_id: 352924, comment_id: 185257 }, {	{ user_id: 1912, question_id: 6483277, accepted_answer_id: 6483299, comment_id: 7621623 }, { user_id: 91, question_id: 90812, accepted_answer_id: 90814, comment_id: 13168 }, { user_id: 1912, question_id: 15115547, accepted_answer_id: 15116273, comment_id: 21285863 }, { user_id: 1912, question_id: 15115547, accepted_answer_id: 15116273, comment_id: 21295382 }, { user_id: 1912, question_id: 15115547, accepted_answer_id: 15116273, comment_id: 21273065 }, { user_id: 22227, question_id: 11726249, accepted_answer_id: 11738004, comment_id: 15642310 }, { user_id: 91, question_id: 1119630, accepted_answer_id: 1119760, comment_id: 937333 }, {

	user_id: 147386, question_id: 1478432, accepted_answer_id: 1484079, comment_id: 1338046 }, { user_id: 113713, question_id: 920606, accepted_answer_id: 926456, comment_id: 742318 }, { user_id: 91, question_id: 4771772, accepted_answer_id: 5011833, comment_id: 5700743 }	user_id: 1912, question_id: 4843013, accepted_answer_id: 4893523, comment_id: 5449335 }, { user_id: 1912, question_id: 4843013, accepted_answer_id: 4893523, comment_id: 5454828 }, { user_id: 1912, question_id: 4843013, accepted_answer_id: 4893523, comment_id: 5457611 }

3: Performance Demonstration

Database	database_A	Database_B
Query *2.i*	dbA.badges.aggregate([{ $match: { name: "Nice Question", date: { $gte: ISODate("2020-01-01"), }, }, }, { $lookup: { from: "users", localField: "user_id", foreignField: "id", as: "user", }, }, { $unwind: "$user", }, { $lookup: { from: "questions", localField: "user.id", foreignField: "owner_user_id", as: "questions", }, }, { $unwind: "$questions", }, { $match: { "questions.accepted_answer_id": { $nin: [NaN, null], }, }, }, { $lookup: { from: "answers", localField: "questions.accepted_answer_id", foreignField: "id", as: "accepted_answer", }, }, { $unwind: { path: "$accepted_answer", preserveNullAndEmptyArrays: true, }, }, {	dbB.badges.aggregate([{ $match: { name: "Nice Question", date: { $gte: ISODate("2020-01-01"), }, }, }, { $lookup: { from: "questions", localField: "user.id", foreignField: "owner_user_id", as: "questions", }, }, { $unwind: "$questions", }, { $match: { "questions.accepted_answer_id": { $nin: [NaN, null], }, }, }, { $project: { _id : 0, user_id: "$user.id", user_display_name: "$user.display_name", badge_name: "$name", badge_date: "$date", question_id: "$questions.id", question_title: "$questions.title", accepted_answer_id: "$questions.accepted_answer_id", answer_score:{$ifNull: ["$questions.accepted_answer.score", null]}, }, },]).explain('executionStats')

	`$project: { _id : 0, user_id: "$user.id", user_display_name: "$user.display_name", badge_name: "$name", badge_date: "$date", question_id: "$questions.id", question_title: "$questions.title", accepted_answer_id: "$questions.accepted_answer_id", answer_score:{$ifNull: ["$accepted_answer.score", null]}, }, },]).explain('executionStats')`	
Output	`{ explainVersion: '2', stages: [{ '$cursor': { queryPlanner: { namespace: 'database_A.badges', indexFilterSet: false, parsedQuery: { '$and': [{ name: { '$eq': 'Nice Question' } }, { date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") } }] }, queryHash: 'B12102D7', planCacheKey: '76B9D3F3', maxIndexedOrSolutionsReached: false, maxIndexedAndSolutionsReached: false, maxScansToExplodeReached: false, winningPlan: { queryPlan: { stage: 'PROJECTION_SIMPLE', planNodeId: 2, transformBy: { date: true, name: true, user_id: true, _id: false }, inputStage: { stage: 'COLLSCAN', planNodeId: 1, filter: { '$and': [{ name: { '$eq': 'Nice Question' } }, { date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") } }] }, direction: 'forward' }`	`{ explainVersion: '2', stages: [{ '$cursor': { queryPlanner: { namespace: 'database_B.badges', indexFilterSet: false, parsedQuery: { '$and': [{ name: { '$eq': 'Nice Question' } }, { date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") } }] }, queryHash: '8130A78A', planCacheKey: '481C7E3A', maxIndexedOrSolutionsReached: false, maxIndexedAndSolutionsReached: false, maxScansToExplodeReached: false, winningPlan: { queryPlan: { stage: 'PROJECTION_DEFAULT', planNodeId: 2, transformBy: { date: true, name: true, user: { display_name: true, id: true }, _id: false }, inputStage: { stage: 'COLLSCAN', planNodeId: 1, filter: { '$and': [{ name: { '$eq': 'Nice Question' } }, { date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") }`

<table>
<tr>
<td></td>
<td>
<pre>
            },
            slotBasedPlan: {
              slots: '$$RESULT=s10 env: {
s3 = 1703682369270 (NOW), s9 =
1577836800000, s1 =
TimeZoneDatabase(Asia/Ho_Chi_Minh...Antarct
ica/Rothera) (timeZoneDB), s2 = Nothing
(SEARCH_META), s8 = "Nice Question" }',
              stages: '[2] mkbson s10 s6
[date, name, user_id] keep [] true false
\n' +
                '[1] filter {(traverseF(s5,
lambda(l1.0) { ((l1.0 == s8) ?: false) },
false) && traverseF(s4, lambda(l2.0) {
((l2.0 >= s9) ?: false) }, false))} \n' +
                '[1] scan s6 s7 none none
none none lowPriority [s4 = date, s5 =
name] @"38f47b88-06a7-432b-8aed-
a45e6720139d" true false '
            }
          },
          rejectedPlans: []
        },
        executionStats: {
          executionSuccess: true,
          nReturned: 19,
          executionTimeMillis: 13370,
          totalKeysExamined: 0,
          totalDocsExamined: 11964,
          executionStages: {
            stage: 'mkbson',
            planNodeId: 2,
            nReturned: 19,
            executionTimeMillisEstimate: 5,
            opens: 1,
            closes: 1,
            saveState: 12,
            restoreState: 12,
            isEOF: 1,
            objSlot: 10,
            rootSlot: 6,
            fieldBehavior: 'keep',
            fields: ['date', 'name',
'user_id'],
            projectFields: [],
            projectSlots: [],
            forceNewObject: true,
            returnOldObject: false,
            inputStage: {
              stage: 'filter',
              planNodeId: 1,
              nReturned: 19,
              executionTimeMillisEstimate:
5,
              opens: 1,
              closes: 1,
              saveState: 12,
              restoreState: 12,
              isEOF: 1,
              numTested: 11964,
</pre>
</td>
<td>
<pre>
                }
              ]
            },
            direction: 'forward'
          }
        },
        slotBasedPlan: {
          slots: '$$RESULT=s10 env: {
s3 = 1703682411208 (NOW), s9 =
1577836800000, s1 =
TimeZoneDatabase(Asia/Ho_Chi_Minh...Antarct
ica/Rothera) (timeZoneDB), s2 = Nothing
(SEARCH_META), s8 = "Nice Question" }',
          stages: '[2] project [s10 =
traverseP(s6, lambda(l3.0) { \n' +
            '   if isObject(l3.0) \n'
+
            '   then
makeBsonObj(MakeObjSpec(keep, ["date",
"name"], ["user"]), l3.0,
traverseP(getField(l3.0, "user"),
lambda(l4.0) { \n' +
            '   if isObject(l4.0)
\n' +
            '   then
makeBsonObj(MakeObjSpec(keep,
["display_name", "id"], []), l4.0) \n' +
            '   else Nothing \n' +
            '   }, Nothing)) \n' +
            '   else Nothing \n' +
            '}, Nothing)] \n' +
            '[1] filter {(traverseF(s5,
lambda(l1.0) { ((l1.0 == s8) ?: false) },
false) && traverseF(s4, lambda(l2.0) {
((l2.0 >= s9) ?: false) }, false))} \n' +
            '[1] scan s6 s7 none none
none none lowPriority [s4 = date, s5 =
name] @"bfcb56f9-5046-4c5d-954d-
8ab48570807a" true false '
          }
        },
        rejectedPlans: []
      },
      executionStats: {
        executionSuccess: true,
        nReturned: 19,
        executionTimeMillis: 89,
        totalKeysExamined: 0,
        totalDocsExamined: 11964,
        executionStages: {
          stage: 'project',
          planNodeId: 2,
          nReturned: 19,
          executionTimeMillisEstimate: 5,
          opens: 1,
          closes: 1,
          saveState: 12,
          restoreState: 12,
          isEOF: 1,
          projections: {
</pre>
</td>
</tr>
</table>

```
              filter: '(traverseF(s5,
lambda(l1.0) { ((l1.0 == s8) ?: false) },
false) && traverseF(s4, lambda(l2.0) {
((l2.0 >= s9) ?: false) }, false)) ',
              inputStage: {
                stage: 'scan',
                planNodeId: 1,
                nReturned: 11964,

executionTimeMillisEstimate: 5,
                opens: 1,
                closes: 1,
                saveState: 12,
                restoreState: 12,
                isEOF: 1,
                numReads: 11964,
                recordSlot: 6,
                recordIdSlot: 7,
                fields: [ 'date', 'name' ],
                outputSlots: [ Long("4"),
Long("5") ]
              }
            }
          }
        }
      },
      nReturned: Long("19"),
      executionTimeMillisEstimate:
Long("5")
    },
    {
      '$lookup': {
        from: 'users',
        as: 'user',
        localField: 'user_id',
        foreignField: 'id',
        unwinding: {
preserveNullAndEmptyArrays: false }
      },
      totalDocsExamined: Long("3800"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("19"),
      indexesUsed: [],
      nReturned: Long("19"),
      executionTimeMillisEstimate:
Long("13")
    },
    {
      '$lookup': {
        from: 'questions',
        as: 'questions',
        localField: 'user.id',
        foreignField: 'owner_user_id',
        let: {},
        pipeline: [
          {
            '$match': {
              accepted_answer_id: { '$not':
{ '$in': [ null, NaN ] } }
            }
```

```
            '10': 'traverseP(s6,
lambda(l3.0) { \n' +
              '    if isObject(l3.0) \n'
+
              '    then
makeBsonObj(MakeObjSpec(keep, ["date",
"name"], ["user"]), l3.0,
traverseP(getField(l3.0, "user"),
lambda(l4.0) { \n' +
              '        if isObject(l4.0)
\n' +
              '        then
makeBsonObj(MakeObjSpec(keep,
["display_name", "id"], []), l4.0) \n' +
              '        else Nothing \n' +
              '    }, Nothing)) \n' +
              '    else Nothing \n' +
              '}, Nothing) '
          },
          inputStage: {
            stage: 'filter',
            planNodeId: 1,
            nReturned: 19,
            executionTimeMillisEstimate:
5,
            opens: 1,
            closes: 1,
            saveState: 12,
            restoreState: 12,
            isEOF: 1,
            numTested: 11964,
            filter: '(traverseF(s5,
lambda(l1.0) { ((l1.0 == s8) ?: false) },
false) && traverseF(s4, lambda(l2.0) {
((l2.0 >= s9) ?: false) }, false)) ',
            inputStage: {
              stage: 'scan',
              planNodeId: 1,
              nReturned: 11964,

executionTimeMillisEstimate: 5,
              opens: 1,
              closes: 1,
              saveState: 12,
              restoreState: 12,
              isEOF: 1,
              numReads: 11964,
              recordSlot: 6,
              recordIdSlot: 7,
              fields: [ 'date', 'name' ],
              outputSlots: [ Long("4"),
Long("5") ]
            }
          }
        }
      },
      nReturned: Long("19"),
      executionTimeMillisEstimate:
Long("5")
```

```
          }
        ],
        unwinding: {
preserveNullAndEmptyArrays: false }
      },
      totalDocsExamined: Long("70623"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("19"),
      indexesUsed: [],
      nReturned: Long("1142"),
      executionTimeMillisEstimate:
Long("82")
    },
    {
      '$lookup': {
        from: 'answers',
        as: 'accepted_answer',
        localField:
'questions.accepted_answer_id',
        foreignField: 'id',
        unwinding: {
preserveNullAndEmptyArrays: true }
      },
      totalDocsExamined: Long("17889430"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("1142"),
      indexesUsed: [],
      nReturned: Long("1142"),
      executionTimeMillisEstimate:
Long("13366")
    },
    {
      '$project': {
        user_id: '$user.id',
        user_display_name:
'$user.display_name',
        badge_name: '$name',
        badge_date: '$date',
        question_id: '$questions.id',
        question_title: '$questions.title',
        accepted_answer_id:
'$questions.accepted_answer_id',
        answer_score: { '$ifNull': [
'$accepted_answer.score', { '$const': null
} ] },
        _id: false
      },
      nReturned: Long("1142"),
      executionTimeMillisEstimate:
Long("13366")
    }
  ],
  serverInfo: {
    host: 'LAPTOP-NF883RJQ',
    port: 27017,
    version: '7.0.2',
    gitVersion:
'02b3c655e1302209ef046da6ba3ef6749dd0b62a'
  },
  serverParameters: {
```

```
    },
    {
      '$lookup': {
        from: 'questions',
        as: 'questions',
        localField: 'user.id',
        foreignField: 'owner_user_id',
        let: {},
        pipeline: [
          {
            '$match': {
              accepted_answer_id: { '$not':
{ '$in': [ null, NaN ] } }
            }
          }
        ],
        unwinding: {
preserveNullAndEmptyArrays: false }
      },
      totalDocsExamined: Long("70623"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("19"),
      indexesUsed: [],
      nReturned: Long("1142"),
      executionTimeMillisEstimate:
Long("87")
    },
    {
      '$project': {
        user_id: '$user.id',
        user_display_name:
'$user.display_name',
        badge_name: '$name',
        badge_date: '$date',
        question_id: '$questions.id',
        question_title: '$questions.title',
        accepted_answer_id:
'$questions.accepted_answer_id',
        answer_score: {
          '$ifNull': [
'$questions.accepted_answer.score', {
'$const': null } ]
        },
        _id: false
      },
      nReturned: Long("1142"),
      executionTimeMillisEstimate:
Long("87")
    }
  ],
  serverInfo: {
    host: 'LAPTOP-NF883RJQ',
    port: 27017,
    version: '7.0.2',
    gitVersion:
'02b3c655e1302209ef046da6ba3ef6749dd0b62a'
  },
  serverParameters: {
    internalQueryFacetBufferSizeBytes:
104857600,
```

	internalQueryFacetBufferSizeBytes: 104857600, internalQueryFacetMaxOutputDocSizeBytes: 104857600, internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600, internalDocumentSourceGroupMaxMemoryBytes: 104857600, internalQueryMaxBlockingSortMemoryUsageBytes: 104857600, internalQueryProhibitBlockingMergeOnMongoS: 0, internalQueryMaxAddToSetBytes: 104857600, internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600, internalQueryFrameworkControl: 'trySbeEngine' }, command: { aggregate: 'badges', pipeline: [{ '$match': { name: 'Nice Question', date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") } } }, { '$lookup': { from: 'users', localField: 'user_id', foreignField: 'id', as: 'user' } }, { '$unwind': '$user' }, { '$lookup': { from: 'questions', localField: 'user.id', foreignField: 'owner_user_id', as: 'questions' } }, { '$unwind': '$questions' }, { '$match': { 'questions.accepted_answer_id': { '$nin': [NaN, null] } } }, { '$lookup': {	internalQueryFacetMaxOutputDocSizeBytes: 104857600, internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600, internalDocumentSourceGroupMaxMemoryBytes: 104857600, internalQueryMaxBlockingSortMemoryUsageBytes: 104857600, internalQueryProhibitBlockingMergeOnMongoS: 0, internalQueryMaxAddToSetBytes: 104857600, internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600, internalQueryFrameworkControl: 'trySbeEngine' }, command: { aggregate: 'badges', pipeline: [{ '$match': { name: 'Nice Question', date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") } } }, { '$lookup': { from: 'questions', localField: 'user.id', foreignField: 'owner_user_id', as: 'questions' } }, { '$unwind': '$questions' }, { '$match': { 'questions.accepted_answer_id': { '$nin': [NaN, null] } } }, { '$project': { _id: 0, user_id: '$user.id', user_display_name: '$user.display_name', badge_name: '$name', badge_date: '$date', question_id: '$questions.id', question_title: '$questions.title', accepted_answer_id: '$questions.accepted_answer_id',

	from: 'answers', localField: 'questions.accepted_answer_id', foreignField: 'id', as: 'accepted_answer' } }, { '$unwind': { path: '$accepted_answer', preserveNullAndEmptyArrays: true } }, { '$project': { _id: 0, user_id: '$user.id', user_display_name: '$user.display_name', badge_name: '$name', badge_date: '$date', question_id: '$questions.id', question_title: '$questions.title', accepted_answer_id: '$questions.accepted_answer_id', answer_score: { '$ifNull': ['$accepted_answer.score', null] } } }], cursor: {}, '$db': 'database_A' }, ok: 1 }	answer_score: { '$ifNull': ['$questions.accepted_answer.score', null] } } }], cursor: {}, '$db': 'database_B' }, ok: 1 }
Query *2.ii*	dbA.badges.aggregate([{ $match: { name: "Nice Question", date: { $gte: ISODate("2020-01-01"), }, }, }, { $lookup: { from: "users", localField: "user_id", foreignField: "id", as: "user", }, }, { $unwind: "$user", }, { $lookup: { from: "questions", localField: "user_id", foreignField: "owner_user_id",	dbB.badges.aggregate([{ $match: { name: "Nice Question", date: { $gte: ISODate("2020-01-01"), }, }, }, { $lookup: { from: "questions", localField: "user.id", foreignField: "owner_user_id", as: "questions", }, }, { $unwind: "$questions", }, { $group: { _id: "$questions.accepted_answer_id", accepted_answer_id: {

```
      as: "questions",
    },
  },
  {
    $unwind: "$questions",
  },
  {
    $group: {
      _id: "$questions.accepted_answer_id",
      accepted_answer_id: {
        $first:
"$questions.accepted_answer_id",
      },
      questions_id: {
        $first: "$questions.id",
      },
      user_id: {
        $first: "$user.id",
      },
    },
  },
  {
    $match: {
      accepted_answer_id: {
        $nin: [NaN, null],
      },
    },
  },
  {
    $lookup: {
      from: "answers",
      localField: "accepted_answer_id",
      foreignField: "id",
      as: "accepted_answer",
    },
  },
  {
    $unwind: {
      path: "$accepted_answer",
      preserveNullAndEmptyArrays: true,
    },
  },
  {
    $lookup: {
      from: "comments",
      localField: "accepted_answer_id",
      foreignField: "post_id",
      as: "accepted_answer_comments",
    },
  },
  {
    $unwind: "$accepted_answer_comments",
  },
  {
    $project: {
      _id: 0,
      user_id: "$user_id",
      question_id: "$questions_id",
      accepted_answer_id: {$ifNull:
["$accepted_answer_id", null]},
```

```
        $first:
"$questions.accepted_answer_id",
      },
      question_id: {
        $first: "$questions.id",
      },
      user_id: {
        $first: "$user.id",
      },
    },
  },
  {
    $match: {
      accepted_answer_id: {
        $nin: [NaN, null],
      },
    },
  },
  {
    $lookup: {
      from: "comments",
      localField: "accepted_answer_id",
      foreignField: "post_id",
      as: "accepted_answer_comments",
    },
  },
  {
    $unwind: "$accepted_answer_comments",
  },
  {
    $project: {
      _id: 0,
      user_id: "$user_id",
      question_id: "$question_id",
      accepted_answer_id: {$ifNull:
["$accepted_answer_id", null]},
      comment_id: {$ifNull:
["$accepted_answer_comments.id", null]},
    },
  },
]).explain('executionStats')
```

	comment_id: {$ifNull: ["$accepted_answer_comments.id", null]}, }, },]).explain('executionStats')	
Output	{ explainVersion: '2', stages: [{ '$cursor': { queryPlanner: { namespace: 'database_A.badges', indexFilterSet: false, parsedQuery: { '$and': [{ name: { '$eq': 'Nice Question' } }, { date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") } }] }, queryHash: '75CA024B', planCacheKey: '378A631F', maxIndexedOrSolutionsReached: false, maxIndexedAndSolutionsReached: false, maxScansToExplodeReached: false, winningPlan: { queryPlan: { stage: 'PROJECTION_SIMPLE', planNodeId: 2, transformBy: { user_id: true, _id: false }, inputStage: { stage: 'COLLSCAN', planNodeId: 1, filter: { '$and': [{ name: { '$eq': 'Nice Question' } }, { date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") } }] }, direction: 'forward' } }, slotBasedPlan: { slots: '$$RESULT=s10 env: { s3 = 1703682451280 (NOW), s9 = 1577836800000, s1 = TimeZoneDatabase(Asia/Ho_Chi_Minh...Antarctica/Rothera) (timeZoneDB), s2 = Nothing (SEARCH_META), s8 = "Nice Question" }', stages: '[2] mkbson s10 s6 [user_id] keep [] true false \n' +	{ explainVersion: '2', stages: [{ '$cursor': { queryPlanner: { namespace: 'database_B.badges', indexFilterSet: false, parsedQuery: { '$and': [{ name: { '$eq': 'Nice Question' } }, { date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") } }] }, queryHash: '786008DD', planCacheKey: '27B8A8C1', maxIndexedOrSolutionsReached: false, maxIndexedAndSolutionsReached: false, maxScansToExplodeReached: false, winningPlan: { queryPlan: { stage: 'PROJECTION_DEFAULT', planNodeId: 2, transformBy: { user: { id: true }, _id: false }, inputStage: { stage: 'COLLSCAN', planNodeId: 1, filter: { '$and': [{ name: { '$eq': 'Nice Question' } }, { date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") } }] }, direction: 'forward' } }, slotBasedPlan: { slots: '$$RESULT=s10 env: { s3 = 1703682536202 (NOW), s9 = 1577836800000, s1 = TimeZoneDatabase(Asia/Ho_Chi_Minh...Antarctica/Rothera) (timeZoneDB), s2 = Nothing (SEARCH_META), s8 = "Nice Question" }', stages: '[2] project [s10 = traverseP(s6, lambda(l3.0) { \n' +

	'[1] filter {(traverseF(s5, lambda(l1.0) { ((l1.0 == s8) ?: false) }, false) && traverseF(s4, lambda(l2.0) { ((l2.0 >= s9) ?: false) }, false))} \n' + '[1] scan s6 s7 none none none none lowPriority [s4 = date, s5 = name] @"38f47b88-06a7-432b-8aed-a45e6720139d" true false ' } }, rejectedPlans: [] }, executionStats: { executionSuccess: true, nReturned: 19, executionTimeMillis: 22042, totalKeysExamined: 0, totalDocsExamined: 11964, executionStages: { stage: 'mkbson', planNodeId: 2, nReturned: 19, executionTimeMillisEstimate: 5, opens: 1, closes: 1, saveState: 12, restoreState: 12, isEOF: 1, objSlot: 10, rootSlot: 6, fieldBehavior: 'keep', fields: ['user_id'], projectFields: [], projectSlots: [], forceNewObject: true, returnOldObject: false, inputStage: { stage: 'filter', planNodeId: 1, nReturned: 19, executionTimeMillisEstimate: 5, opens: 1, closes: 1, saveState: 12, restoreState: 12, isEOF: 1, numTested: 11964, filter: '(traverseF(s5, lambda(l1.0) { ((l1.0 == s8) ?: false) }, false) && traverseF(s4, lambda(l2.0) { ((l2.0 >= s9) ?: false) }, false)) ', inputStage: { stage: 'scan', planNodeId: 1, nReturned: 11964, executionTimeMillisEstimate: 5, opens: 1, closes: 1,	' if isObject(l3.0) \n' + ' then makeBsonObj(MakeObjSpec(keep, [], ["user"]), l3.0, traverseP(getField(l3.0, "user"), lambda(l4.0) { \n' + ' if isObject(l4.0) \n' + ' then makeBsonObj(MakeObjSpec(keep, ["id"], []), l4.0) \n' + ' else Nothing \n' + ' }, Nothing)) \n' + ' else Nothing \n' + '}, Nothing)] \n' + '[1] filter {(traverseF(s5, lambda(l1.0) { ((l1.0 == s8) ?: false) }, false) && traverseF(s4, lambda(l2.0) { ((l2.0 >= s9) ?: false) }, false))} \n' + '[1] scan s6 s7 none none none none lowPriority [s4 = date, s5 = name] @"bfcb56f9-5046-4c5d-954d-8ab48570807a" true false ' } }, rejectedPlans: [] }, executionStats: { executionSuccess: true, nReturned: 19, executionTimeMillis: 14681, totalKeysExamined: 0, totalDocsExamined: 11964, executionStages: { stage: 'project', planNodeId: 2, nReturned: 19, executionTimeMillisEstimate: 6, opens: 1, closes: 1, saveState: 12, restoreState: 12, isEOF: 1, projections: { '10': 'traverseP(s6, lambda(l3.0) { \n' + ' if isObject(l3.0) \n' + ' then makeBsonObj(MakeObjSpec(keep, [], ["user"]), l3.0, traverseP(getField(l3.0, "user"), lambda(l4.0) { \n' + ' if isObject(l4.0) \n' + ' then makeBsonObj(MakeObjSpec(keep, ["id"], []), l4.0) \n' + ' else Nothing \n' + ' }, Nothing)) \n' + ' else Nothing \n' +

```
                saveState: 12,
                restoreState: 12,
                isEOF: 1,
                numReads: 11964,
                recordSlot: 6,
                recordIdSlot: 7,
                fields: [ 'date', 'name' ],
                outputSlots: [ Long("4"),
Long("5") ]
              }
            }
          }
        }
      },
      nReturned: Long("19"),
      executionTimeMillisEstimate:
Long("5")
    },
    {
      '$lookup': {
        from: 'users',
        as: 'user',
        localField: 'user_id',
        foreignField: 'id',
        unwinding: {
preserveNullAndEmptyArrays: false }
      },
      totalDocsExamined: Long("4000"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("20"),
      indexesUsed: [],
      nReturned: Long("19"),
      executionTimeMillisEstimate:
Long("15")
    },
    {
      '$lookup': {
        from: 'questions',
        as: 'questions',
        localField: 'user_id',
        foreignField: 'owner_user_id',
        unwinding: {
preserveNullAndEmptyArrays: false }
      },
      totalDocsExamined: Long("74340"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("20"),
      indexesUsed: [],
      nReturned: Long("1401"),
      executionTimeMillisEstimate:
Long("76")
    },
    {
      '$group': {
        _id:
'$questions.accepted_answer_id',
        accepted_answer_id: { '$first':
'$questions.accepted_answer_id' },
        questions_id: { '$first':
'$questions.id' },
```

```
                '}, Nothing) '
            },
            inputStage: {
              stage: 'filter',
              planNodeId: 1,
              nReturned: 19,
              executionTimeMillisEstimate:
6,
              opens: 1,
              closes: 1,
              saveState: 12,
              restoreState: 12,
              isEOF: 1,
              numTested: 11964,
              filter: '(traverseF(s5,
lambda(l1.0) { ((l1.0 == s8) ?: false) },
false) && traverseF(s4, lambda(l2.0) {
((l2.0 >= s9) ?: false) }, false)) ',
              inputStage: {
                stage: 'scan',
                planNodeId: 1,
                nReturned: 11964,

executionTimeMillisEstimate: 6,
                opens: 1,
                closes: 1,
                saveState: 12,
                restoreState: 12,
                isEOF: 1,
                numReads: 11964,
                recordSlot: 6,
                recordIdSlot: 7,
                fields: [ 'date', 'name' ],
                outputSlots: [ Long("4"),
Long("5") ]
              }
            }
          }
        }
      },
      nReturned: Long("19"),
      executionTimeMillisEstimate:
Long("6")
    },
    {
      '$lookup': {
        from: 'questions',
        as: 'questions',
        localField: 'user.id',
        foreignField: 'owner_user_id',
        unwinding: {
preserveNullAndEmptyArrays: false }
      },
      totalDocsExamined: Long("74340"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("20"),
      indexesUsed: [],
      nReturned: Long("1401"),
      executionTimeMillisEstimate:
Long("78")
```

```
        user_id: { '$first': '$user.id' }
      },
      maxAccumulatorMemoryUsageBytes: {
        accepted_answer_id: Long("45864"),
        questions_id: Long("45864"),
        user_id: Long("45864")
      },
      totalOutputDataSizeBytes:
Long("253071"),
      usedDisk: false,
      spills: Long("0"),
      spilledDataStorageSize: Long("0"),
      numBytesSpilledEstimate: Long("0"),
      spilledRecords: Long("0"),
      nReturned: Long("819"),
      executionTimeMillisEstimate:
Long("76")
    },
    {
      '$match': {
        accepted_answer_id: { '$not': {
'$in': [ null, NaN ] } }
      },
      nReturned: Long("818"),
      executionTimeMillisEstimate:
Long("76")
    },
    {
      '$lookup': {
        from: 'answers',
        as: 'accepted_answer',
        localField: 'accepted_answer_id',
        foreignField: 'id',
        unwinding: {
preserveNullAndEmptyArrays: true }
      },
      totalDocsExamined: Long("12813970"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("818"),
      indexesUsed: [],
      nReturned: Long("818"),
      executionTimeMillisEstimate:
Long("9618")
    },
    {
      '$lookup': {
        from: 'comments',
        as: 'accepted_answer_comments',
        localField: 'accepted_answer_id',
        foreignField: 'post_id',
        unwinding: {
preserveNullAndEmptyArrays: false }
      },
      totalDocsExamined: Long("18223404"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("818"),
      indexesUsed: [],
      nReturned: Long("375"),
      executionTimeMillisEstimate:
Long("22042")
    },
    {
      '$group': {
        _id:
'$questions.accepted_answer_id',
        accepted_answer_id: { '$first':
'$questions.accepted_answer_id' },
        question_id: { '$first':
'$questions.id' },
        user_id: { '$first': '$user.id' }
      },
      maxAccumulatorMemoryUsageBytes: {
        accepted_answer_id: Long("45864"),
        question_id: Long("45864"),
        user_id: Long("45864")
      },
      totalOutputDataSizeBytes:
Long("253071"),
      usedDisk: false,
      spills: Long("0"),
      spilledDataStorageSize: Long("0"),
      numBytesSpilledEstimate: Long("0"),
      spilledRecords: Long("0"),
      nReturned: Long("819"),
      executionTimeMillisEstimate:
Long("79")
    },
    {
      '$match': {
        accepted_answer_id: { '$not': {
'$in': [ null, NaN ] } }
      },
      nReturned: Long("818"),
      executionTimeMillisEstimate:
Long("79")
    },
    {
      '$lookup': {
        from: 'comments',
        as: 'accepted_answer_comments',
        localField: 'accepted_answer_id',
        foreignField: 'post_id',
        unwinding: {
preserveNullAndEmptyArrays: false }
      },
      totalDocsExamined: Long("18223404"),
      totalKeysExamined: Long("0"),
      collectionScans: Long("818"),
      indexesUsed: [],
      nReturned: Long("375"),
      executionTimeMillisEstimate:
Long("14680")
    },
    {
      '$project': {
        user_id: '$user_id',
        question_id: '$question_id',
        accepted_answer_id: { '$ifNull': [
'$accepted_answer_id', { '$const': null } ]
},
```

	`},` `{` `'$project': {` `user_id: '$user_id',` `question_id: '$questions_id',` `accepted_answer_id: { '$ifNull': [ '$accepted_answer_id', { '$const': null } ] },` `comment_id: {` `'$ifNull': [ '$accepted_answer_comments.id', { '$const': null } ]` `},` `_id: false` `},` `nReturned: Long("375"),` `executionTimeMillisEstimate: Long("22042")` `}` `],` `serverInfo: {` `host: 'LAPTOP-NF883RJQ',` `port: 27017,` `version: '7.0.2',` `gitVersion: '02b3c655e1302209ef046da6ba3ef6749dd0b62a'` `},` `serverParameters: {` `internalQueryFacetBufferSizeBytes: 104857600,` `internalQueryFacetMaxOutputDocSizeBytes: 104857600,` `internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,` `internalDocumentSourceGroupMaxMemoryBytes: 104857600,` `internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,` `internalQueryProhibitBlockingMergeOnMongoS: 0,` `internalQueryMaxAddToSetBytes: 104857600,` `internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,` `internalQueryFrameworkControl: 'trySbeEngine'` `},` `command: {` `aggregate: 'badges',` `pipeline: [` `{` `'$match': {` `name: 'Nice Question',`	`comment_id: {` `'$ifNull': [ '$accepted_answer_comments.id', { '$const': null } ]` `},` `_id: false` `},` `nReturned: Long("375"),` `executionTimeMillisEstimate: Long("14680")` `}` `],` `serverInfo: {` `host: 'LAPTOP-NF883RJQ',` `port: 27017,` `version: '7.0.2',` `gitVersion: '02b3c655e1302209ef046da6ba3ef6749dd0b62a'` `},` `serverParameters: {` `internalQueryFacetBufferSizeBytes: 104857600,` `internalQueryFacetMaxOutputDocSizeBytes: 104857600,` `internalLookupStageIntermediateDocumentMaxSizeBytes: 104857600,` `internalDocumentSourceGroupMaxMemoryBytes: 104857600,` `internalQueryMaxBlockingSortMemoryUsageBytes: 104857600,` `internalQueryProhibitBlockingMergeOnMongoS: 0,` `internalQueryMaxAddToSetBytes: 104857600,` `internalDocumentSourceSetWindowFieldsMaxMemoryBytes: 104857600,` `internalQueryFrameworkControl: 'trySbeEngine'` `},` `command: {` `aggregate: 'badges',` `pipeline: [` `{` `'$match': {` `name: 'Nice Question',` `date: { '$gte': ISODate("2020-01-01T00:00:00.000Z") }` `}` `},` `{` `'$lookup': {` `from: 'questions',` `localField: 'user.id',` `foreignField: 'owner_user_id',`

```
          date: { '$gte': ISODate("2020-01-
01T00:00:00.000Z") }
        }
      },
      {
        '$lookup': {
          from: 'users',
          localField: 'user_id',
          foreignField: 'id',
          as: 'user'
        }
      },
      { '$unwind': '$user' },
      {
        '$lookup': {
          from: 'questions',
          localField: 'user_id',
          foreignField: 'owner_user_id',
          as: 'questions'
        }
      },
      { '$unwind': '$questions' },
      {
        '$group': {
          _id:
'$questions.accepted_answer_id',
          accepted_answer_id: { '$first':
'$questions.accepted_answer_id' },
          questions_id: { '$first':
'$questions.id' },
          user_id: { '$first': '$user.id' }
        }
      },
      {
        '$match': { accepted_answer_id: {
'$nin': [ NaN, null ] } }
      },
      {
        '$lookup': {
          from: 'answers',
          localField: 'accepted_answer_id',
          foreignField: 'id',
          as: 'accepted_answer'
        }
      },
      {
        '$unwind': { path:
'$accepted_answer',
preserveNullAndEmptyArrays: true }
      },
      {
        '$lookup': {
          from: 'comments',
          localField: 'accepted_answer_id',
          foreignField: 'post_id',
          as: 'accepted_answer_comments'
        }
      },
      { '$unwind':
'$accepted_answer_comments' },
          as: 'questions'
        }
      },
      { '$unwind': '$questions' },
      {
        '$group': {
          _id:
'$questions.accepted_answer_id',
          accepted_answer_id: { '$first':
'$questions.accepted_answer_id' },
          question_id: { '$first':
'$questions.id' },
          user_id: { '$first': '$user.id' }
        }
      },
      {
        '$match': { accepted_answer_id: {
'$nin': [ NaN, null ] } }
      },
      {
        '$lookup': {
          from: 'comments',
          localField: 'accepted_answer_id',
          foreignField: 'post_id',
          as: 'accepted_answer_comments'
        }
      },
      { '$unwind':
'$accepted_answer_comments' },
      {
        '$project': {
          _id: 0,
          user_id: '$user_id',
          question_id: '$question_id',
          accepted_answer_id: { '$ifNull':
[ '$accepted_answer_id', null ] },
          comment_id: { '$ifNull': [
'$accepted_answer_comments.id', null ] }
        }
      }
    ],
    cursor: {},
    '$db': 'database_B'
  },
  ok: 1
}
```

```
      {
        '$project': {
          _id: 0,
          user_id: '$user_id',
          question_id: '$questions_id',
          accepted_answer_id: { '$ifNull':
[ '$accepted_answer_id', null ] },
          comment_id: { '$ifNull': [
'$accepted_answer_comments.id', null ] }
        }
      }
    ],
    cursor: {},
    '$db': 'database_A'
  },
  ok: 1
}
```

To demonstrate the performance of each query, the method `.explain('executionStats')` was attached to the end of each query (usage highlighted in yellow). This method prints statistics that describe the execution of the query. The particular statistic that is most relevant to the query performance is *executionTimeMillis*, as this displays the total time required to execute the query in milliseconds (highlighted in yellow).

For both queries *database_B* outperformed *database_A*. In the case of the query used for Question *2.i*, *database_B* executed its query in 89 milliseconds whilst *database_A* executed its query in 13370 milliseconds. In the case of the query used for Question *2.ii*, *database_B* executed its query in 14681 milliseconds whilst *database_A* executed its query in 22042 milliseconds.

4: Discussion

Suitability of the Embedded Model for *Stack Overflow*

The embedded design demonstrated in *database_B* is likely not suitable for the *Stack Overflow* full application in regards to scalability though there are a mixture of advantages and disadvantages that the embedded design would have.

Advantages of the Embedded Design

Join Operations

The embedded design reduces the need for join operations by embedding documents in fields that are likely to be involved in popular queries. For example the field *accepted_answer* containing an *Answer* document can be found inside

a *Question* collection as it is likely that finding the accepted answer for a question is a popular query, this particular case would save the need for a join operation between the *accepted_answer_id* field in the *Question* collection and the *id* field in the *Answer* collection. Join operations are time-consuming especially with a large quantity of rows as described in the case of the *Stack Overflow* database with 11 million *Users*, 18 million *Questions*, 27 million *Answers*, and 75 million *Comments*. The reduced need for joins becomes more significant of an advantage with scale as joins become more costly with scale. Using an embedded design is especially useful for data that is read-heavy and has relatively static relationships between entities. Unfortunately this is not likely to be the case with the *Stack Overflow* database.

Flaws of the Embedded Design

The main flaws of using the embedded design for the *Stack Overflow* database is the growth in individual document size with scale and the potential updates to embedded documents needed with scale.

Document Size

In the case of the *Stack Overflow* database, as it contains so many documents initially, the embedding of multiple other documents into a single document can double or triple the size of each document. For example, in *database_B*, a *Question* can contain an *Answer* document inside the field *accepted_answer*, and a *User* document in both the *last_editor* and *owner* fields. Whilst not every *Question* has an *accepted_answer*, it needs to contain a *last_editor* and *owner* document, and as each entity is of a similar size (a *Question* is of a similar size to a *User*) this is likely to triple the size of each *Question*. The size of each document becomes a larger issue as the database grows in quantity, as inefficiencies in document size are reflected more significantly in larger-scale databases such as the *Stack Overflow* database. This can lead to heavy performance issues as larger documents consume more memory and can slow down query processing.

Updating Fields

Another issue with the embedded model presented in *database_B* is that updates to a single document such as *User* can create a situation where updates are needed across many other documents in the database. For example, if a *User* changes their *about_me* information, every single *Question*, *Comment*, *Badge*,

and *Answer* associated with that user will also need to update their embedded user document. This problem also grows in significance with scale as the more documents there are, the more need to be searched to see if they contain the document in need of updating. In the *Stack Overflow* case, 18 million *Questions*, 27 million *Answers*, and 75 million *Comments* would need to be searched to check if they contain the user who hypothetically changed their *about_me* section.

Sharding Difficulties

An embedded model is also challenging to horizontally scale as the relationships between documents are often complex and difficult to shard. For example, in the *Answers* collection, which embeds documents like *parent_question*, *last_editor*, and *owner*. When sharding, you might choose a shard key like the answer ID. However, this doesn't account for the embedded documents, which may have their own relationships and distribution needs. For instance, if *parent_question* is embedded within *Answers*, sharding by answer ID could separate a question from its answers if they reside on different shards. This separation complicates queries that need to access a question and its associated answers together, leading to potential performance issues and complexities in maintaining data integrity across shards.

Conclusion

These factors make *database_B* unsuitable due to the lack of scalability that the model has and whilst the lack of need for joins can help increase performance it is unlikely to compensate for the other two issues caused by the embedded design on larger-scale databases like the *Stack Overflow* database. The difficulty to horizontally scale also adds to the lack of scalability.

Alternative Model

A more suitable modelling option for scalability would be a *Key-Value* database model combination with sharding. A *Key-Value* database model would simply use the *id* field for each entity as the key and the other fields in each entity as values.

Advantages of a Key-Value Model

Document Size and Updating Values

In a *Key-Value* database there will be a more efficient use of storage compared to the embedded model as seen in *database_B* as there are no embedded documents, therefore the size of a document/row is limited to only the size of itself. In addition, the issue relating to updating embedded documents in *database_B* is not present as related documents/rows are represented by an id field containing a key and not by embedding. This means that when a related document/row is changed, this change only needs to be made once as opposed to potentially multiple times as in *database_B*.

Sharding

The *Key-Value* model can be easily horizontally scaled across multiple shards. In this case hash-based sharding can be applied. In a hash-based sharding system the shard key is passed through a hash function which determines the shard that the data is stored and accessed from.

The shard key can simply be the *Answer*, *Badge*, *Comment*, *Question*, or *User* ID, applied with a hash-based sharding system for the following reasons:

- High cardinality – each ID is unique, offering a vast range of distinct values for sharding.
- Low frequency – while IDs are unique and have low frequency, the hash function applied to these IDs ensures a more uniform distribution of data across shards, preventing hotspots.
- Non-monotonically changing – whilst the IDs are monotonically changing (assuming they are assigned sequentially), hash-based sharding can be applied.

Sharding helps the scalability of this solution as it allows the database to handle more data and traffic by spreading the load of data across multiple shards.

Disadvantages of a Key-Value Model

Join Operations

Unfortunately the *Key-Value* model requires more join operations when handling joining data from multiple keys which is expensive and inefficient, in particular over a large scale. Embedded models do not have this inefficiency as, if joining data is needed from different collections, a document is embedded and

can be easily accessed in the same way a field is accessed. As a *Key-Value* database scales up the amount of interrelated data increases which can be particularly challenging in scenarios where relationships between different data entities are crucial for functionality, as it requires additional logic in the application layer to manually handle these relationships, thereby increasing the complexity and potentially affecting performance. For example, if we wanted to find all the *Users* who created *Comments* for a particular *Question*, a join operation would have to be performed between the *id* field of the *Question* and the *post_id* for the *Comment*, the a join would need to be made between the *user_id* of the *Comment* and all *id* fields of *Users*.

Conclusion

Whilst the Key-Value database model presents challenges with join operations, its advantages in terms of scalability make it more suitable for a large-scale application like *Stack Overflow*. The ease of sharding and horizontal scaling, combined with a lower overall data size and simplified update processes, effectively outweighs the drawbacks when comparing to the embedded model in *database_B* for the increasing *Stack Overflow* database.

Question 2: Neo4j

1: Writing Queries

i)

Query

```
MATCH (u:User)-[r:EARNED]->(b:Badge {name: 'Fanatic'})
RETURN u.userId AS user_id, u.display_name AS display_name, b.name
AS badge_name, b.date AS date_awarded
```

Answers (limit 10 records)


```
[
  {
    "user_id": 91,
    "display_name": "Michael Stum",
    "badge_name": "Fanatic",
    "date_awarded": 1252078069000
  },
  {
    "user_id": 19719,
    "display_name": "indiv",
    "badge_name": "Fanatic",
    "date_awarded": 1329838464000
  },
  {
    "user_id": 27190,
    "display_name": "martijno",
    "badge_name": "Fanatic",
    "date_awarded": 1363326328000
  },
  {
    "user_id": 32775,
    "display_name": "luiscubal",
    "badge_name": "Fanatic",
    "date_awarded": 1359851241000
  },
  {
    "user_id": 47453,
    "display_name": "Bill Lynch",
    "badge_name": "Fanatic",
    "date_awarded": 1287547086000
  },
  {
    "user_id": 50924,
    "display_name": "Dan McNevin",
    "badge_name": "Fanatic",
    "date_awarded": 1284080538000
  },
  {
    "user_id": 86646,
    "display_name": "Eric Dahlvang",
    "badge_name": "Fanatic",
    "date_awarded": 1337051341000
  },
```

```
  {
    "user_id": 104815,
    "display_name": "Steve",
    "badge_name": "Fanatic",
    "date_awarded": 1282010262000
  },
  {
    "user_id": 115555,
    "display_name": "John Slavick",
    "badge_name": "Fanatic",
    "date_awarded": 1417133404000
  },
  {
    "user_id": 136208,
    "display_name": "AProgrammer",
    "badge_name": "Fanatic",
    "date_awarded": 1300960012000
  }
]
```


ii)

Query

```
MATCH (u:User)-[r:EARNED]->(b:Badge {name: 'Nice Question'})
WHERE b.date >= apoc.date.parse('2020-01-01', 'ms', 'yyyy-MM-dd')
MATCH (u)-[:POSTS]->(q:Question)
RETURN u.userId AS UserID,
       u.display_name AS DisplayName,
       b.name AS BadgeName,
       b.date AS BadgeAwardedDate,
       q.questionId AS QuestionID,
       q.title AS QuestionTitle
```

Answers (limit 10 records)


```
[
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 277926,
    "QuestionTitle": "Is there a Maximum Length for
userPrincipalName in Active Directory?"
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 1146044,
    "QuestionTitle": "Join-Free Table structure for Tags"
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
```

```
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 190516,
    "QuestionTitle": "Getting all direct Reports from Active
Directory"
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 10405583,
    "QuestionTitle": "Cannot convert X to T even though X should
match T?"
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 2605056,
    "QuestionTitle": "While loop in IL - why stloc.0 and ldloc.0?"
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 2483,
    "QuestionTitle": "Casting: (NewType) vs. Object as NewType"
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 1418277,
    "QuestionTitle": "Controller/Static State Class in WinForms
Application - Where to put?"
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 1114691,
    "QuestionTitle": "Relationships and Lazy Loading in SubSonic
3.0"
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 35219,
    "QuestionTitle": "Optimizing/Customizing Sharepoint Search
Crawling"
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
```

```
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 1836805,
    "QuestionTitle": "Empty Visual Studio Project?"
  }
]
```


iii)

Query

```
MATCH (u:User)-[:EARNED]->(b:Badge {name: 'Nice Question'})
WHERE b.date >= apoc.date.parse('2020-01-01', 'ms', 'yyyy-MM-dd')
MATCH (u)-[:POSTS]->(q:Question)
MATCH (q)-[:HAS_COMMENT]->(c:Comment)
RETURN u.userId AS UserID,
       u.display_name AS DisplayName,
       b.name AS BadgeName,
       b.date AS BadgeAwardedDate,
       q.questionId AS QuestionID,
       q.title AS QuestionTitle,
       c.commentId AS CommentID
```

Answers (limit 10 records)


```
[
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 1146044,
    "QuestionTitle": "Join-Free Table structure for Tags",
    "CommentID": 1160640
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 190516,
    "QuestionTitle": "Getting all direct Reports from Active
Directory",
    "CommentID": 69746
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 10405583,
    "QuestionTitle": "Cannot convert X to T even though X should
match T?",
    "CommentID": 13423479
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
```

```
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 2605056,
    "QuestionTitle": "While loop in IL - why stloc.0 and
ldloc.0?",
    "CommentID": 2615159
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 2483,
    "QuestionTitle": "Casting: (NewType) vs. Object as NewType",
    "CommentID": 28516542
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 1836805,
    "QuestionTitle": "Empty Visual Studio Project?",
    "CommentID": 1729877
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 1836805,
    "QuestionTitle": "Empty Visual Studio Project?",
    "CommentID": 1729467
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 608851,
    "QuestionTitle": "What goes on a WebFrontend and what on the
Application Server is Sharepoint 2007/WSS 3.0?",
    "CommentID": 466017
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 4026413,
    "QuestionTitle": "Is a Session-Less Design feasible?",
    "CommentID": 4318777
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Nice Question",
    "BadgeAwardedDate": 1583097901000,
    "QuestionID": 4026413,
    "QuestionTitle": "Is a Session-Less Design feasible?",
    "CommentID": 4319407
```

```
  }
]
```

iv)

Query

```
MATCH (u:User)-[:EARNED]->(b:Badge {name: 'Inquisitive'})
MATCH (u)-[:POSTS]->(q:Question)
WHERE q.accepted_answer_id IS NOT NULL
RETURN u.userId AS UserID,
       u.display_name AS DisplayName,
       b.name AS BadgeName,
       q.questionId AS QuestionID,
       q.title AS QuestionTitle,
       q.accepted_answer_id AS AcceptedAnswerID
```

Answers (limit 10 records)


```
[
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
    "QuestionID": 277926,
    "QuestionTitle": "Is there a Maximum Length for
userPrincipalName in Active Directory?",
    "AcceptedAnswerID": 323081
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
    "QuestionID": 1146044,
    "QuestionTitle": "Join-Free Table structure for Tags",
    "AcceptedAnswerID": 1146060
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
    "QuestionID": 190516,
    "QuestionTitle": "Getting all direct Reports from Active
Directory",
    "AcceptedAnswerID": 190570
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
    "QuestionID": 10405583,
    "QuestionTitle": "Cannot convert X to T even though X should
match T?",
    "AcceptedAnswerID": 10405614
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
```

```
    "QuestionID": 2605056,
    "QuestionTitle": "While loop in IL - why stloc.0 and
ldloc.0?",
    "AcceptedAnswerID": 2605058
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
    "QuestionID": 2483,
    "QuestionTitle": "Casting: (NewType) vs. Object as NewType",
    "AcceptedAnswerID": 2487
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
    "QuestionID": 1418277,
    "QuestionTitle": "Controller/Static State Class in WinForms
Application - Where to put?",
    "AcceptedAnswerID": 1418355
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
    "QuestionID": 1114691,
    "QuestionTitle": "Relationships and Lazy Loading in SubSonic
3.0",
    "AcceptedAnswerID": 1115463
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
    "QuestionID": 35219,
    "QuestionTitle": "Optimizing/Customizing Sharepoint Search
Crawling",
    "AcceptedAnswerID": 38231
  },
  {
    "UserID": 91,
    "DisplayName": "Michael Stum",
    "BadgeName": "Inquisitive",
    "QuestionID": 1836805,
    "QuestionTitle": "Empty Visual Studio Project?",
    "AcceptedAnswerID": 1836850
  }
]
```


2: Query Performance

Query 1: *iv*

Query Selection

```
PROFILE MATCH (u:User)-[:EARNED]->(b:Badge {name: 'Inquisitive'})
MATCH (u)-[:POSTS]->(q:Question)
```

```
WHERE q.accepted_answer_id IS NOT NULL
RETURN u.userId AS UserID,
       u.display_name AS DisplayName,
       b.name AS BadgeName,
       q.questionId AS QuestionID,
       q.title AS QuestionTitle,
       q.accepted_answer_id AS AcceptedAnswerID
```

The query from part *iv* is used with the "PROFILE" clause in front (highlighted in yellow). This generates an execution plan for the query and executes the plan, providing statistics such as rows passed on at each stage, database hits at each stage, elapsed time and memory usage.

Results


NodeIndexSeek@neo4j
b
RANGE INDEX b:Badge(name) WHERE name = $autostring_0, cache[b.name]
248memory (bytes)
2,845pagecache hits
0pagecache misses
74estimated rows
21db hits
20rows
Expand(All)@neo4j
b, anon_0, u
(b)<-[anon_0:EARNED]-(u)
74estimated rows
40db hits
20rows
Filter@neo4j
b, anon_0, u
u:User
74estimated rows
104db hits
20rows
CacheProperties@neo4j
b, anon_0, u
cache[u.display_name], cache[u.userId]
74estimated rows
156db hits
20rows
Expand(All)@neo4j
u, anon_0, anon_1, q, b
(u)-[anon_1:POSTS]->(q)
1,373estimated rows
2,148db hits
2,048rows
Filter@neo4j
u, anon_0, anon_1, q, b
cache[q.accepted_answer_id] IS NOT NULL AND q:Question
1,036estimated rows
6,974db hits
1,642rows
CacheProperties@neo4j
u, anon_0, anon_1, q, b
cache[q.questionId], cache[q.title]
1,036estimated rows
4,926db hits
1,642rows
Projection@neo4j
QuestionID, AcceptedAnswerID, UserID, DisplayName, QuestionTitle, BadgeName
cache[u.userId] AS UserID, cache[q.title] AS QuestionTitle, cache[u.display_name] AS DisplayName, cache[q.accepted_answer_id] AS AcceptedAnswerID, cache[b.name] AS BadgeName, cache[q.questionId] AS QuestionID
1,036estimated rows
0db hits
1,642rows
ProduceResults@neo4j
QuestionID, AcceptedAnswerID, UserID, DisplayName, QuestionTitle, BadgeName
UserID, DisplayName, BadgeName, QuestionID, QuestionTitle, AcceptedAnswerID
336total memory (bytes)
0memory (bytes)
1,036estimated rows
0db hits
1,642rows
Result

The following table has been produced. It is suggested that the query should be rerun several times before taking an average, therefore, it was chosen to rerun the query 10 times before taking the average of the next 3 readings. This means that the query had been run a total of 13 times, the initial time time taken on the first run is recorded in the table, then the first 3 times taken after the initial 10 runs are recorded in the table (run 11, 12, 13).

Total DB Hits	14369	*Total Memory (bytes)*	336
Rows Processed	7054		
Elapsed Time			
Attempts		*Time (ms)*	
Initial		60	
1		6	
2		5	
3		7	
Average		6	

Description

See section *Indexed Database* under *Analysis* under *Indexing*.

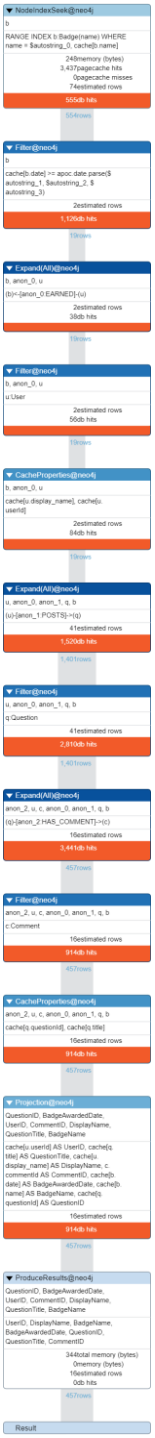
Query 2: *iii*

Query Selection

```
PROFILE MATCH (u:User)-[:EARNED]->(b:Badge {name: 'Nice
Question'})
WHERE b.date >= apoc.date.parse('2020-01-01', 'ms', 'yyyy-MM-dd')
MATCH (u)-[:POSTS]->(q:Question)
MATCH (q)-[:HAS_COMMENT]->(c:Comment)
RETURN u.userId AS UserID,
       u.display_name AS DisplayName,
       b.name AS BadgeName,
       b.date AS BadgeAwardedDate,
       q.questionId AS QuestionID,
       q.title AS QuestionTitle,
       c.commentId AS CommentID
```

The same "PROFILE" clause is inserted into the query from part *iii* to show the execution plan and execution statistics.

Results



The following table has been produced. It is suggested that the query should be rerun several times before taking an average, therefore, it was chosen to rerun the query 10 times before taking the average of the next 3 readings. This means

that the query had been run a total of 13 times, the initial time time taken on the first run is recorded in the table, then the first 3 times taken after the initial 10 runs are recorded in the table (run 11, 12, 13).

Total DB Hits	12372	*Total Memory (bytes)*	344
Rows Processed	5717		
Elapsed Time			
Attempts		*Time (ms)*	
Initial		86	
1		8	
2		6	
3		6	
Average		6.67	

Description

We can observe that there are some stages of efficiency and some stages that are more resource intensive in the query. Overall the results show that the query has a short average execution time of 6.67ms and takes up a small amount of memory (344 bytes). It processed 5717 rows and took a total of 12372 database hits. The query was executed multiple times, with an initial attempt taking 86 milliseconds and subsequent attempts averaging around 6.67 milliseconds. This decrease in execution time is due to caching or optimization after the first run. Overall, the query seems efficient in terms of memory usage, but the number of database hits might be an area for potential optimization.

The query is executed by gathering "Nice Question" *Badges*, then filtering for *Badges* that were awarded after the 1st January 2020. Then *EARNED* relationships between those *Badges* and *Users* are expanded. The associated *Users* are then filtered out and their *display_name* and *userId* are cached for later use. Then the *POSTS* relationships between those *Users* and the associated *Questions* are expanded, at which point the *Questions* are filtered out. After this the *HAS_COMMENT* relationship between those *Questions* and the associated *Comments* are expanded and the *Comments* are filtered out. Then the *questionId* and *title* properties are cached. After, the relevant data is projected.

Initially the database starts off by accessed the index on the *name* field on *Badges* and passing on all *Badges* where the *name* field is "Nice Question" to be used as the anchor point, in this case 554 rows are passed on. This is a point

where the query is performed efficiently because it directly accesses nodes with the “Nice Question” badge name, using an index specifically built for the 'name' property of badges. This indexed search is faster than scanning all badge nodes, as it narrows down the search to only those nodes that match the criterion, significantly reducing the number of database hits and computational effort required. It also immediately narrows down the dataset to only relevant nodes, reducing the amount of data that subsequent steps need to process. This initial filtering limits the scope of the query right from the start, making the following operations more focused and quicker, as they operate on a smaller, more relevant subset of data.

Filters in this query are applied as early as possible to help reduce the processing of unwanted data, helping the execution take place more effectively. Whilst there are relationship traversals which are resource intensive, these happen only when needed and with data that is already filtered, helping the efficiency.

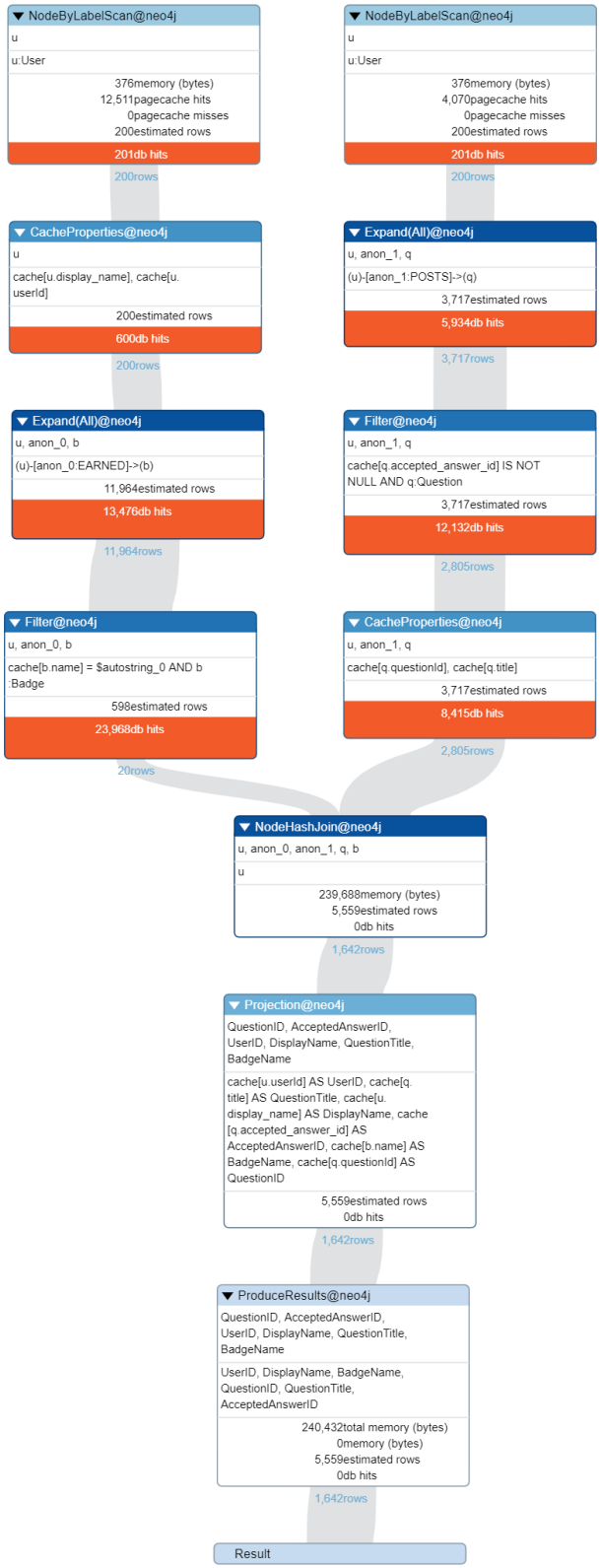
Indexing

Query Selection

```
// Drop all indexes

DROP INDEX user_id_index;
DROP INDEX user_display_name_index;

DROP INDEX badge_name_index;
DROP INDEX badge_date_index;

DROP INDEX question_id_index;
DROP INDEX question_title_index;
DROP INDEX question_accepted_answer_id_index;

DROP INDEX comment_id_index;
```

```
// Profile Query 1

PROFILE MATCH (u:User)-[:EARNED]->(b:Badge {name: 'Inquisitive'})
MATCH (u)-[:POSTS]->(q:Question)
WHERE q.accepted_answer_id IS NOT NULL
RETURN u.userId AS UserID,
       u.display_name AS DisplayName,
       b.name AS BadgeName,
       q.questionId AS QuestionID,
       q.title AS QuestionTitle,
       q.accepted_answer_id AS AcceptedAnswerID
```

Results


NodeByLabelScan@neo4j
u
u:User
376memory (bytes)
12,511pagecache hits
0pagecache misses
200estimated rows
201db hits
200rows
NodeByLabelScan@neo4j
u
u:User
376memory (bytes)
4,070pagecache hits
0pagecache misses
200estimated rows
201db hits
200rows
CacheProperties@neo4j
u
cache[u.display_name], cache[u.userId]
200estimated rows
600db hits
200rows
Expand(All)@neo4j
u, anon_1, q
(u)-[anon_1:POSTS]->(q)
3,717estimated rows
5,934db hits
3,717rows
Expand(All)@neo4j
u, anon_0, b
(u)-[anon_0:EARNED]->(b)
11,964estimated rows
13,476db hits
11,964rows
Filter@neo4j
u, anon_1, q
cache[q.accepted_answer_id] IS NOT NULL AND q:Question
3,717estimated rows
12,132db hits
2,805rows
Filter@neo4j
u, anon_0, b
cache[b.name] = $autostring_0 AND b:Badge
598estimated rows
23,968db hits
20rows
CacheProperties@neo4j
u, anon_1, q
cache[q.questionId], cache[q.title]
3,717estimated rows
8,415db hits
2,805rows
NodeHashJoin@neo4j
u, anon_0, anon_1, q, b
u
239,688memory (bytes)
5,559estimated rows
0db hits
1,642rows
Projection@neo4j
QuestionID, AcceptedAnswerID, UserID, DisplayName, QuestionTitle, BadgeName
cache[u.userId] AS UserID, cache[q.title] AS QuestionTitle, cache[u.display_name] AS DisplayName, cache[q.accepted_answer_id] AS AcceptedAnswerID, cache[b.name] AS BadgeName, cache[q.questionId] AS QuestionID
5,559estimated rows
0db hits
1,642rows
ProduceResults@neo4j
QuestionID, AcceptedAnswerID, UserID, DisplayName, QuestionTitle, BadgeName
UserID, DisplayName, BadgeName, QuestionID, QuestionTitle, AcceptedAnswerID
240,432total memory (bytes)
0memory (bytes)
5,559estimated rows
0db hits
1,642rows
Result

The following table has been produced. It is suggested that the query should be rerun several times before taking an average, therefore, it was chosen to rerun the query 10 times before taking the average of the next 3 readings. This means that the query had been run a total of 13 times, the initial time time taken on the first run is recorded in the table, then the first 3 times taken after the initial 10 runs are recorded in the table (run 11, 12, 13).

Total DB Hits	64927	*Total Memory (bytes)*	240432
Rows Processed	26837		
Elapsed Time			
Attempts		*Time (ms)*	
Initial		73	
1		19	
2		24	
3		22	
Average		21.67	

Analysis

When comparing the non-indexed query performance with indexed it appears that indexes significantly increase the performance of the database. The indexed version has significantly less DB hits (14369 vs 64927), rows processed (7054 vs 26837), memory used (336 bytes vs 240432 bytes), as well as a lower average elapsed time (6ms vs 21.67ms).

The basic differences in structure are that the indexed version starts off using *Badges* which are already filtered for the "Inquisitive" *name* as an anchor using an index and then performs the relevant filtering and retrieval. The non-indexed version runs two parallel branches which both use all *Users* as anchors and filter out *Users* with the relevant badge on one branch, and *Users* with posts on the other before performing a join.

Indexed Database

When performing the query, the indexed database starts by accessing index that contains the *name* of the *Badge* nodes, and selecting the ones that have a *name* matching "Inquisitive" as the anchor, which returns 21 hits and passes on 20 rows to the next stage. Then, all *EARNED* relationships from *Users* with these *Badge* nodes are expanded, returning 40 hits and passing 20 rows to the next stage where *Users* are filtered for, returing 40 hits and passing 20 rows to

the next stage. Next, the *User's display_ name* and *userId* is cached for later use generating 156 hits and passing 20 rows onwards. At this point, of the passed on users, the *Users* who have a *POSTS* relationship with a *Question* have their relationship expanded, generating 2148 hits and passing on 2048 rows. These rows are then filtered to only include *Questions* that have an *accepted_ answer_ id* that is not a null value, generating 6974 hits and passing on 1642 rows. The final stage before projection is simply caching the *questionId* and *title* for the remaining *Questions* generating 4926 hits and passing 1642 rows onwards.

Non-Indexed Database

In stark contrast, the non-indexed version takes the *Users* as an anchor point for one branch, returning 201 hits and passing 200 rows to the next stage which is far less than the indexed versions anchor at 21 hits and 20 rows passed on. It then caches the properties *display_ name* and *userId* which generates 600 hits and passes 200 rows on, which is more costly than in the indexed database (156 hits, 20 rows) as the non-indexed version caches this data before filtering the *Users* to see if they have the "Inquisitive" *Badge*, therefore caching some unused data. It then expands all *EARNED* relationships between all *Users* with *Badge* nodes. This is different from the indexed database which expands the *EARNED* relationships after it has selected the *Badges* that have a *name* matching "Inquisitive" therefore having to expand less relationships, therefore pass less rows (74 vs 11964) and generate less hits (40 vs 13476). After this the non-indexed database filters the *Badges* with the *name* "Inquisitive" which generates 23968 hits and passes on 20 rows.

For the other branch, the non-indexed version also uses *Users* as the anchor point, returning 201 hits and passing on 200 rows. It then expands all *POSTS* relationships between all *Users* and *Questions*. As no sort of filtering has been applied to the *Users* this means that this stage is more costly than the equivalent operation that was done by the indexed database (3717 vs 2048 rows passed on and 5934 vs 2148 hits taken), which had already narrowed down the *Users* being passed on to only those who had earned the "Inquisitive" *Badge*. At this point the *Questions* that have an *accepted_ answer_ id* that is not a null value are filtered out and passed on to the next stage, which generates 12132 hits and passes on a total of 2805 rows. This stage is also more costly than the equivalent

stage in the indexed database (6974 hits and 1642 rows passed on) as more rows are being filtered (2048 in the indexed database vs 3717 in the non-indexed database). The same behaviour is also seen in the next stage where the *questionId* and *title* are cached returning 8415 hits (vs 4926 found in the indexed version) and passing on 2805 rows (vs 1642 found in the indexed version).

After both branches which are running in parallel are executed a join operation is performed on the *Users*, *Badges*, and *Questions*, returning 1642 rows and generating no DB hits. At which point the projection happens.

Conclusion

Whilst the non-indexed database runs two costly branches in parallel, which reduces the time taken to perform the query, it still appears to be far slower than the indexed database. This is probably because each stage is simply far more costly for the non-indexed database, meaning that the slowest branch will be far slower than the entire linear plan performed by the indexed database. The most important factor seems to be that, because the indexed database contains an index on the *name* property for the *Badges*, an initial "filter" can be applied to the *Badges* selecting only the *Badges* that have the *name* "Inquisitive" as an anchor point. This means that the indexed database starts with an anchor point of only 20 rows, which is quite small. This has a cascading effect on every other operation performed as each stage has to deal with proportionately less rows. The non-indexed database does not have access to this index therefore is forced to use the entire *Users* collection as an anchor point which is 200 rows, 10 times larger. It chooses to use the *Users* collection as it is the smallest viable collection to start with as there are only 200 *Users* compared to 3717 *Questions* and 11964 *Badges*.

3: COMSC Stackoverflow Redesign

Updated Model

A new *Collection* node will need to be created.

Collection		
Field	Datatype	Explanation
collectionId	Int32	Collection's unique ID

Name	String	Collection's display name
description	String	Extra information about the collection
post_count	Int32	Number of posts in the collection
creation_date	Date	Creation of the collection
last_edited_date	Date	Last time there was an edit made to the collection
owner_user_id	Int32	User ID of the user who owns the collection

The *Questions* table will need to have a field appended to it.

Question		
Field	Datatype	Explanation
collection_id	Int32	Collection(s) associated with the question

Similarly the *Answers* table will need to have a field appended to it.

Answer		
Field	Datatype	Explanation
collection_id	Int32	Collection(s) associated with the answer

The following relationships will need to be added:

- *User* -> OWNER_OF -> *Collection*
- *Collection* -> HAS_QUESTION -> *Question*
- *Collection* -> CONTAINS_ANSWER -> *Answer*

Queries

Constraints

```
// create constraints
CREATE CONSTRAINT FOR (c:Collection) REQUIRE c.id IS UNIQUE;
```

Example Nodes

```
// create example collection node
CREATE (c:Collection {
    collectionId: toInteger(1),
    name: "Example Collection",
    description: "Questions that belong to the example
collection",
    post_count: toInteger(1),
    creation_date: apoc.date.currentTimestamp(),
```

```
    last_edited_date: apoc.date.currentTimestamp(),
    owner_user_id: toInteger(0)
})
RETURN c;
```

```
// create example question node that is in a collection
CREATE (q:Question {
              questionId: toInteger(0),
              accepted_answer_id: null,
              answer_count: toInteger(0),
              comment_count: toInteger(0),
              creation_date: apoc.date.currentTimestamp(),
              last_activity_date: apoc.date.currentTimestamp(),
              owner_user_id: null,
              post_type: null,
              score: toInteger(0),
              tags: null,
              title: "Example Question",
              view_count: toInteger(0),
              collection_id: [toInteger(1)]
})
RETURN q;
```

```
// create example user node that owns a collection
CREATE (u:User {
              userId: toInteger(0),
              about_me: "I am an example user",
              age: apoc.date.currentTimestamp(),
              creation_date: apoc.date.currentTimestamp(),
              display_name: "example_user",
              down_votes: toInteger(0),
              last_access_date: apoc.date.currentTimestamp(),
              location: null,
              image: null,
              reputation: toInteger(0),
              up_votes: toInteger(0),
              views: toInteger(0),
              website: null
})
RETURN u;
```

```
// create example answer node that is in a collection
CREATE (a:Answer {
       answerId: toInteger(0),
       comment_count: toInteger(0),
       creation_date: apoc.date.currentTimestamp(),
       last_activity_date: apoc.date.currentTimestamp(),
       owner_user_id: toInteger(0),
       parent_id: toInteger(0),
       post_type_id: toInteger(2),
       score: toInteger(0),
       collection_id: [toInteger(1)]
})
RETURN a;
```

Relationships

```
MATCH (u:User), (c:Collection)
WHERE u.userId = c.owner_user_id
MERGE (u)-[:OWNER_OF]->(c)
```

```
MATCH (c:Collection), (q:Question)
WHERE c.collectionId IN q.collection_id
MERGE (c)-[:HAS_QUESTION]->(q)
```

```
MATCH (c:Collection), (a:Answer)
WHERE c.collectionId IN a.collection_id
MERGE (c)-[:CONTAINS_ANSWER]->(a)
```

Relationship Diagrams