Mini Project 4

By: Anirudh Chaudhary, Quan Le, Bob LeBow, Richard More, Hao Nguyen

# Purpose 1

**A - Identify the reviewer with the oldest review overall and within a category.**

Overall

db.reviews.aggregate([{$sort: {unixReviewTime: 1}} ,

{$project: {\_id: 0, reviewerID: 1, reviewerName: 1, reviewTime: 1}},

{$limit: 1}],

{allowDiskUse:true})

Per category

db.reviews.aggregate([{$sort: {"unixReviewTime": 1}},

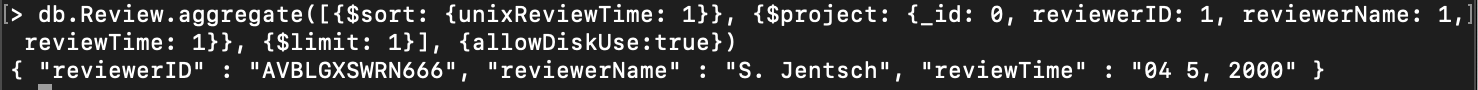
{$group: {\_id: "$category", review: {$first: "$$ROOT"}}},

{$project: {\_id: 0, category: "$\_id", reviewerID: "$review.reviewerID", reviewerName: "$review.reviewerName", reviewTime: "$review.reviewTime"}}],

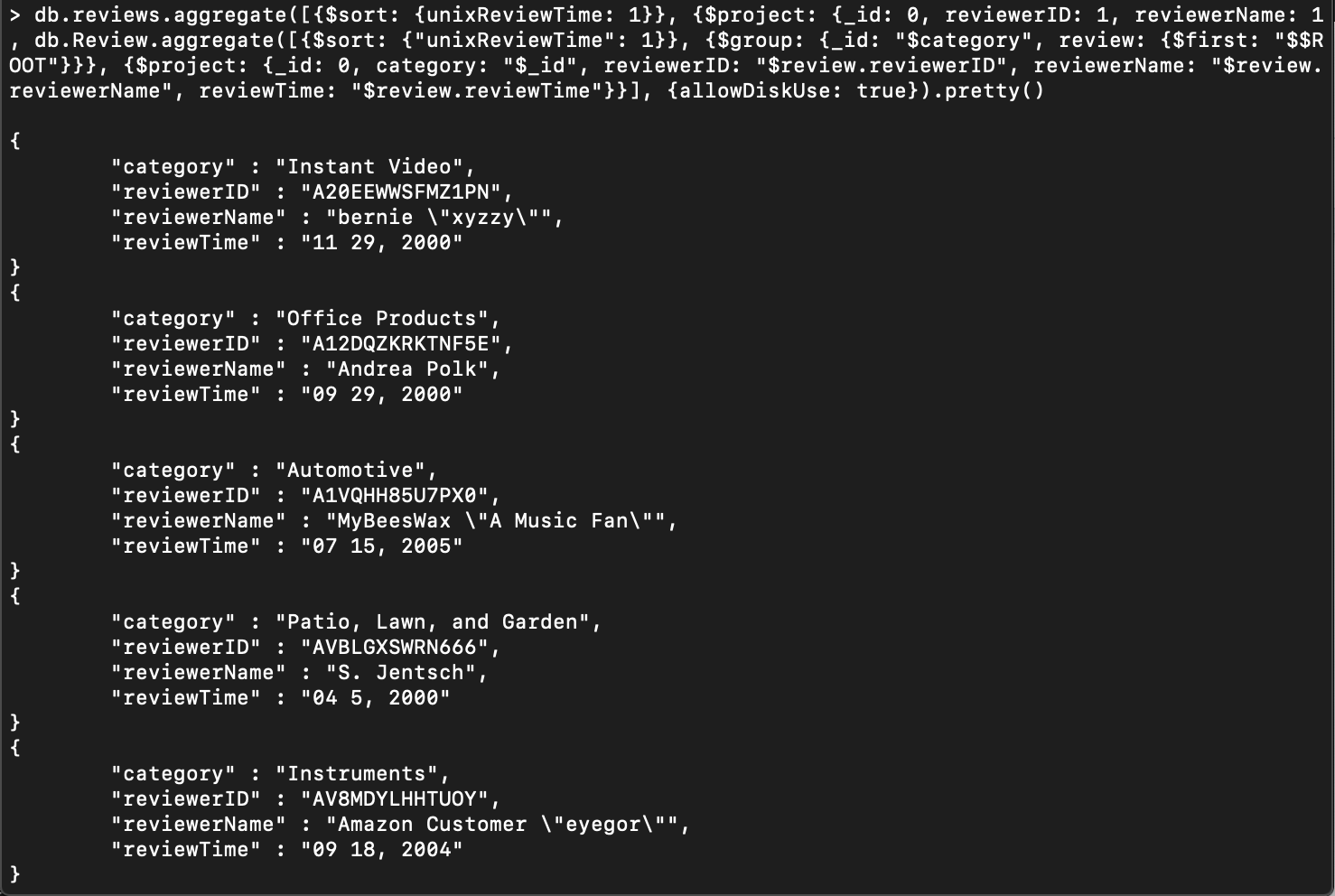
{allowDiskUse: true}).pretty()

*— Snapshot:*

Overall



Per category



*— Summary:*

Overall

The oldest review is made by S. Jensch with reviewerID AVBLGXSWRN666 on April 5, 2000.

Per category

For Instant Video, the oldest review is made by bernie "xyzzy” with reviewerID A20EEWWSFMZ1PN on November 29, 2000.

For Office Products, the oldest review is made by Andrea Polk with reviewerID A12DQZKRKTNF5E on September 29, 2000.

For Automotive, the oldest review is made by MyBeesWax "A Music Fan" with reviewerID A1VQHH85U7PX0 on June 15, 2005.

For Patio, Lawn, and Garden, the oldest review is made by S. Jensch with reviewerID AVBLGXSWRN666 on April 5, 2000.

For Instruments, the oldest review is made by Amazon Customer "eyegor" with reviewerID AV8MDYLHHTUOY on September 18, 2004.

**B - Identify the reviewer with the newest review overall and within a category.**

Overall

db.reviews.aggregate([{$sort: {unixReviewTime: -1}},

{$project: {\_id: 0, reviewerID: 1, reviewerName: 1, reviewTime: 1}},

{$limit: 1}],

{allowDiskUse:true})

Per category

db.reviews.aggregate([{$sort: {"unixReviewTime": -1}},

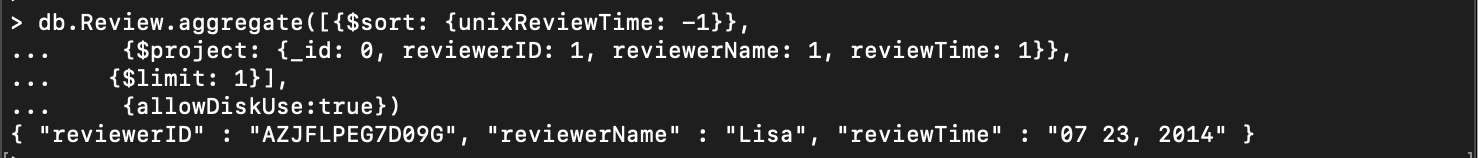
{$group: {\_id: "$category", review: {$first: "$$ROOT"}}},

{$project: {\_id: 0, category: "$\_id", reviewerID: "$review.reviewerID", reviewerName: "$review.reviewerName", reviewTime: "$review.reviewTime"}}],

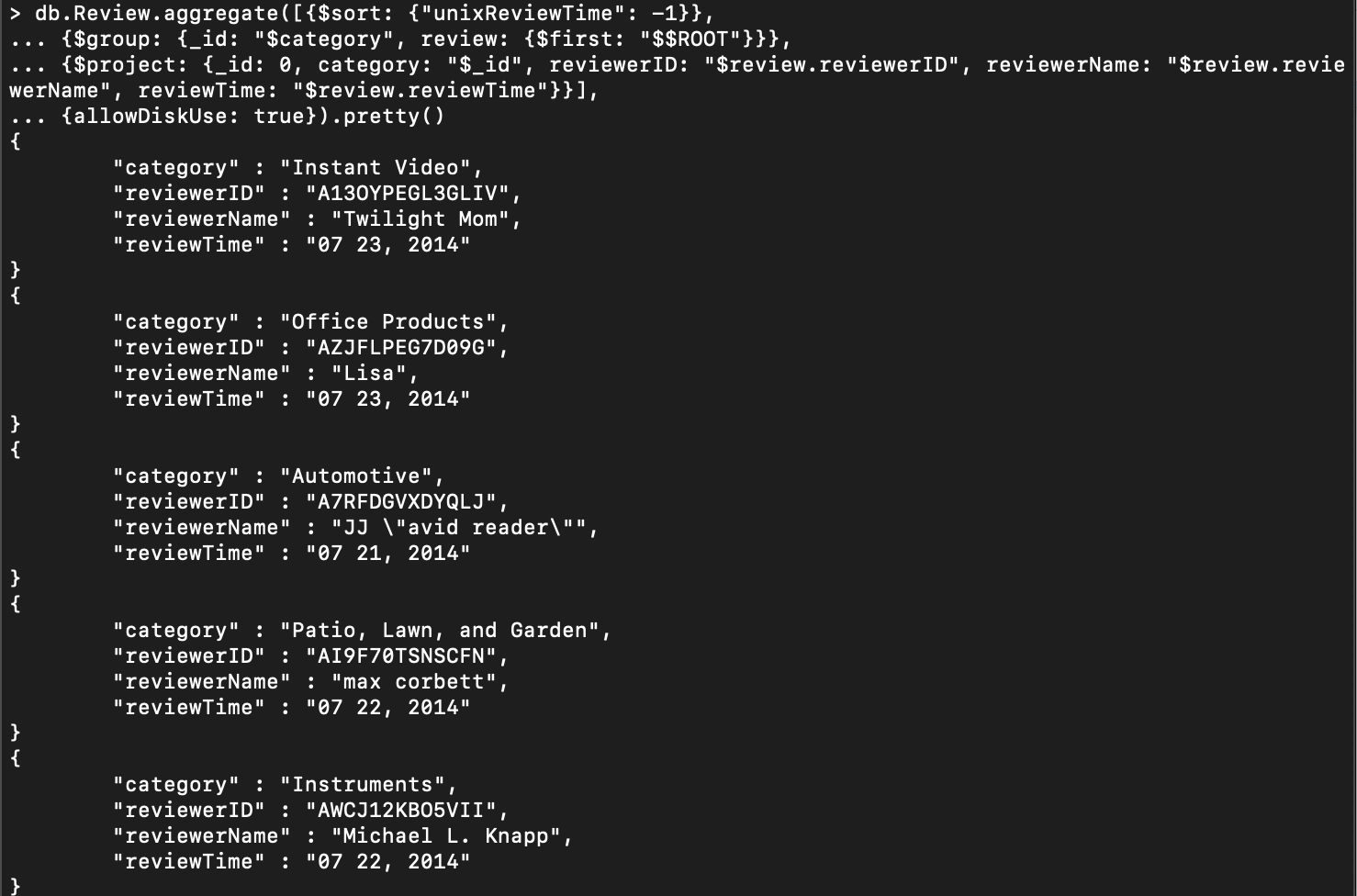
{allowDiskUse: true}).pretty()

*— Snapshot:*

Overall



Per category



*— Summary:*

Overall

The newest reviewer is made by Lisa with reviewerID AZJFLPEG7D09G on June 23, 2014.

Per category

For Instant Video, the newest review is made by Twilight Mom with reviewerID A13OYPEGL3GLIV on July 23, 2014.

For Office Products, the newest review is made by Lisa with reviewerID AZJFLPEG7D09G on July 23, 2014.

For Automotive, the newest review is made by JJ "avid reader" with reviewerID A7RFDGVXDYQLJ on July 21, 2014.

For Patio, Lawn, and Garden, the newest review is made by max corbett with reviewerID AI9F70TSNSCFN on July 22, 2014.

For Instruments, the newest review is made by Michael L. Knapp with reviewerID AWCJ12KBO5VII on July 22, 2014.

**C - Identify the reviewer who has been reviewing for the longest period of time.**

db.reviews.aggregate([{$sort: {"unixReviewTime": 1}},

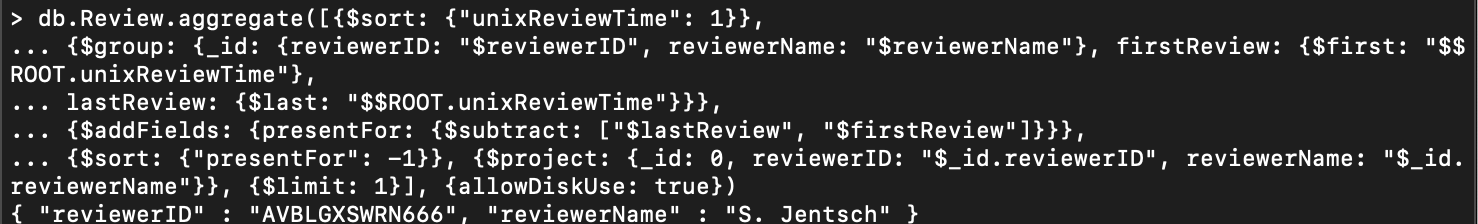
{$group: {\_id: {reviewerID: "$reviewerID", reviewerName: "$reviewerName"}, firstReview: {$first: "$$ROOT.unixReviewTime"},

lastReview: {$last: "$$ROOT.unixReviewTime"}}},

{$addFields: {presentFor: {$subtract: ["$lastReview", "$firstReview"]}}},

{$sort: {"presentFor": -1}}, {$project: {\_id: 0, reviewerID: "$\_id.reviewerID", reviewerName: "$\_id.reviewerName"}}, {$limit: 1}], {allowDiskUse: true})

*— Snapshot:*



*— Summary:*

S. Jentsch with reviewerID AVBLGXSWRN666 is the review who has been reviewing for the longest period of time. The query checks for the difference in the latest review’s date to the oldest review’s date, and it takes the largest difference and projects the the reviewer’s ID and name.

**D - Identify the reviewer whose reviews deviate the most (largest standard deviation of review rating) overall and within a category.**

Overall

db.reviews.aggregate([

{$group: {\_id: {reviewerID: "$reviewerID", reviewerName: "$reviewerName"}, stdev: {$stdDevPop: "$overall"}}},

{$sort: {"stdev": -1}}, {$project: {\_id: 0, reviewerID: "$\_id.reviewerID", reviewerName: "$\_id.reviewerName"}},

{$limit: 1}])

Per category

db.reviews.aggregate([

{$group: {\_id: {

category: "$category",

reviewerID: "$reviewerID",

reviewerName: "$reviewerName"},

stdev: {$stdDevPop: "$overall"}}},

{$sort: {"stdev": -1}},

{$group: {\_id: "$\_id.category",

reviewers: {$push: "$$ROOT"}}},

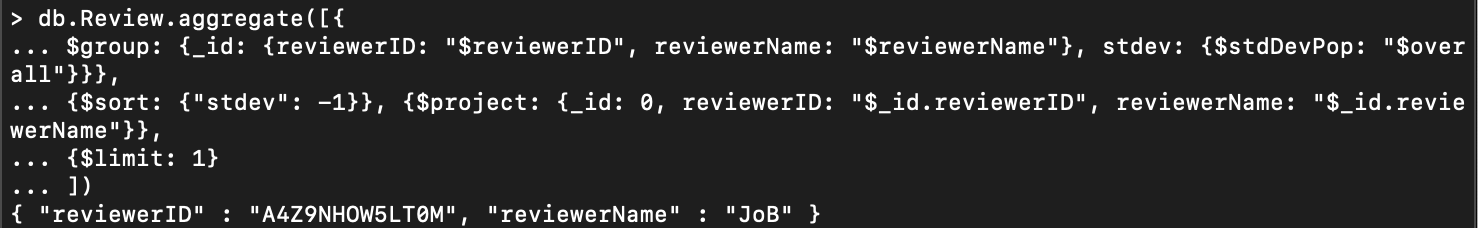
{$project: {\_id: 0, category: "$\_id", reviews: {$arrayElemAt: ["$reviewers", 0]}}},

{$project: {\_id: 0, category: 1, reviewerID: "$reviews.\_id.reviewerID",

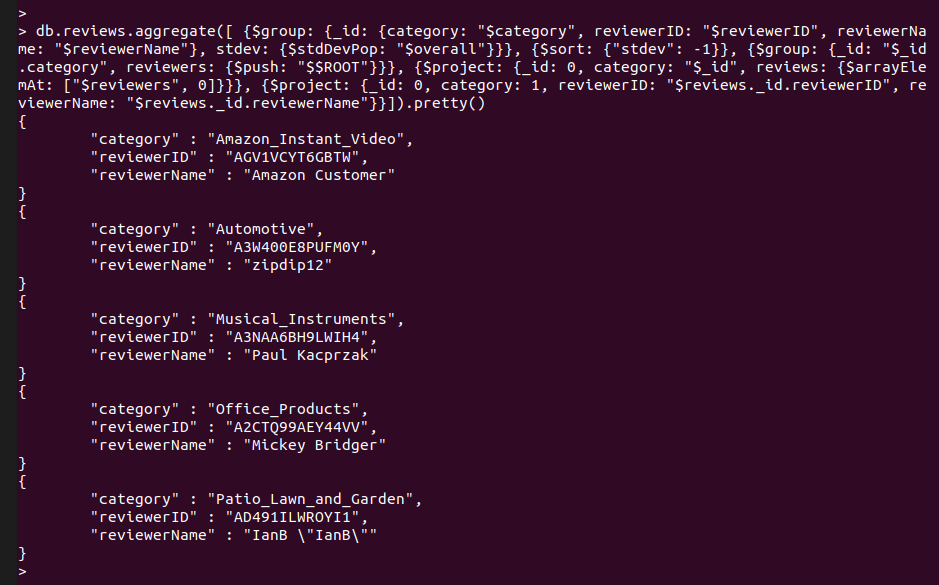
reviewerName: "$reviews.\_id.reviewerName"}}]).pretty()

*— Snapshot:*

Overall

**

Per category



*— Summary:*

*Overall*

The reviewer “JoB” with reviewerID “A4Z9NHOW5LT0M” has the most deviation in the reviews in the overall segment. The query groups the data on ‘reviewerID’ and ‘reviewerName’, then finds the standard deviation as per the ‘overall’ field. Finally sorting the results in descending order as per standard deviation and limiting the results to 1 to project the reviewerID and reviewerName of the one with the highest standard deviation.

*Per category*

The query first groups the data on ‘reviewerID’, ‘reviewerName’ and ‘category’ by calculating the standard deviation of the ‘overall’ field and sorting the data in descending order as per their standard deviation. Then finally grouping the data on the ‘category and reviewers and finally projecting the ‘reviewerID’, ‘reviewerName’ of the reviewer with the highest standard deviation in the respective five categories listed along with the results.

**E - Identify the product with the most divergence in the ratings of reviews received (largest deviation of rating).**

db.reviews.aggregate([

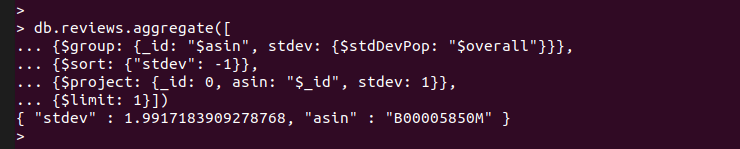
{$group: {\_id: "$asin", stdev: {$stdDevPop: "$overall"}}},

{$sort: {"stdev": -1}},

{$project: {\_id: 0, asin: "$\_id", stdev: 1}},

{$limit: 1}])

*— Snapshot:*



*— Summary:*

The query groups the data on the basis of product id i.e. ‘asin’ and the standard deviation of the ‘overall’ field. Then sorting the data in descending order as per the standard deviation of the ‘overall’ field and limiting the data to 1 in order to project the product id (‘asin’) and standard deviation of the of the product with the largest deviation in rating.

**F - How would you identify the reviewers that reviewed a common set of products?**

#### Solution 1

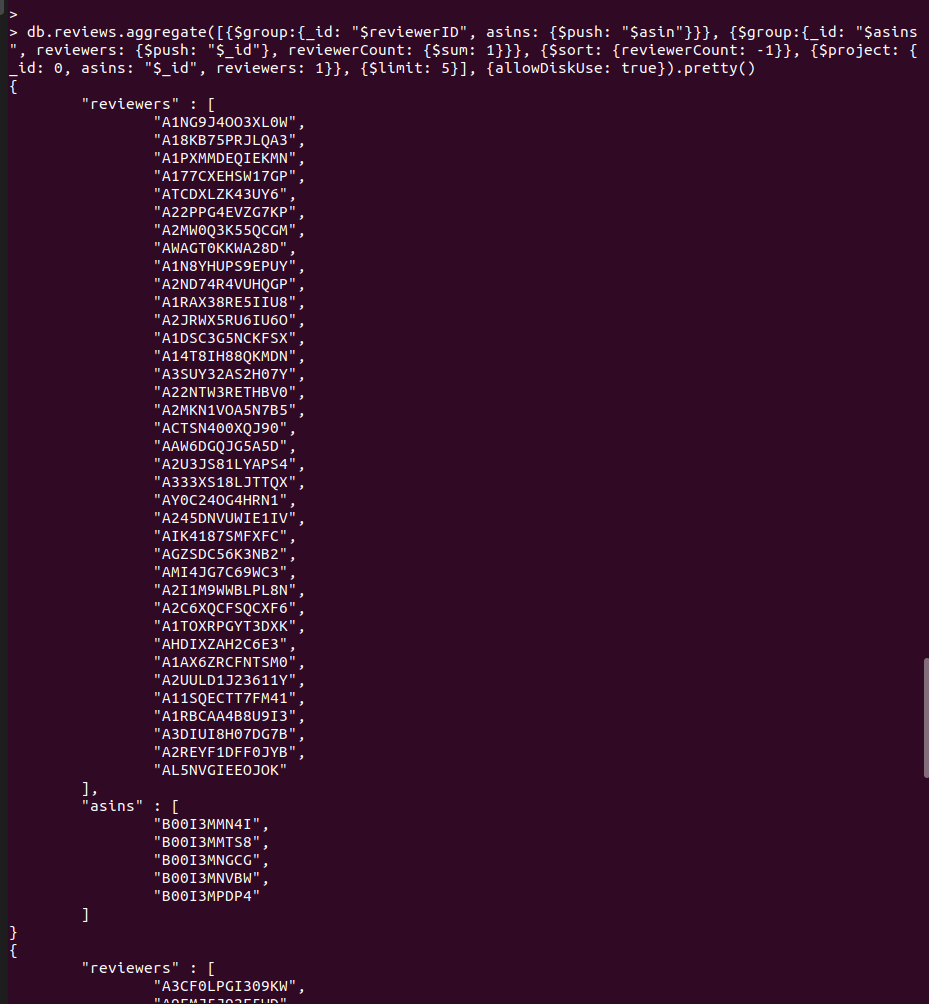
*Wording:* List the reviewers who reviewed the same combination of products

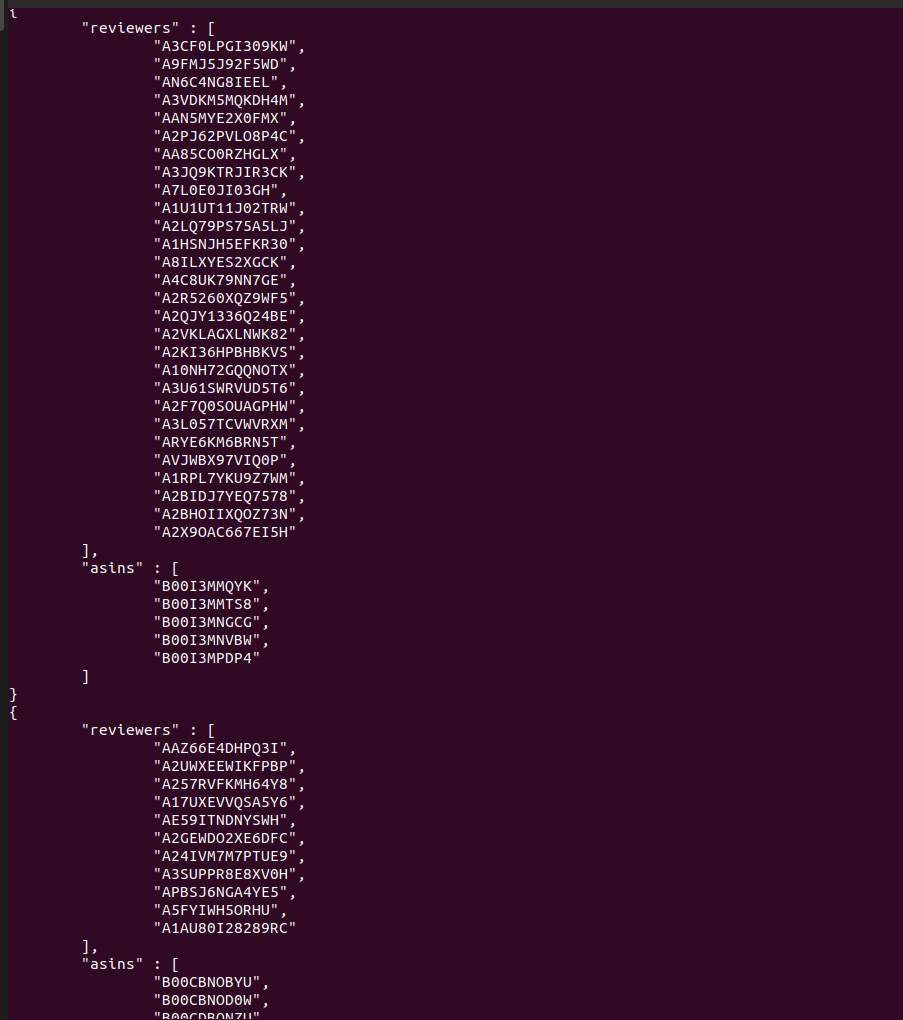
*Limitation:* If for a set of products there are reviewers who reviewed some other products AS WELL then they will not be included in that combination of product IDs

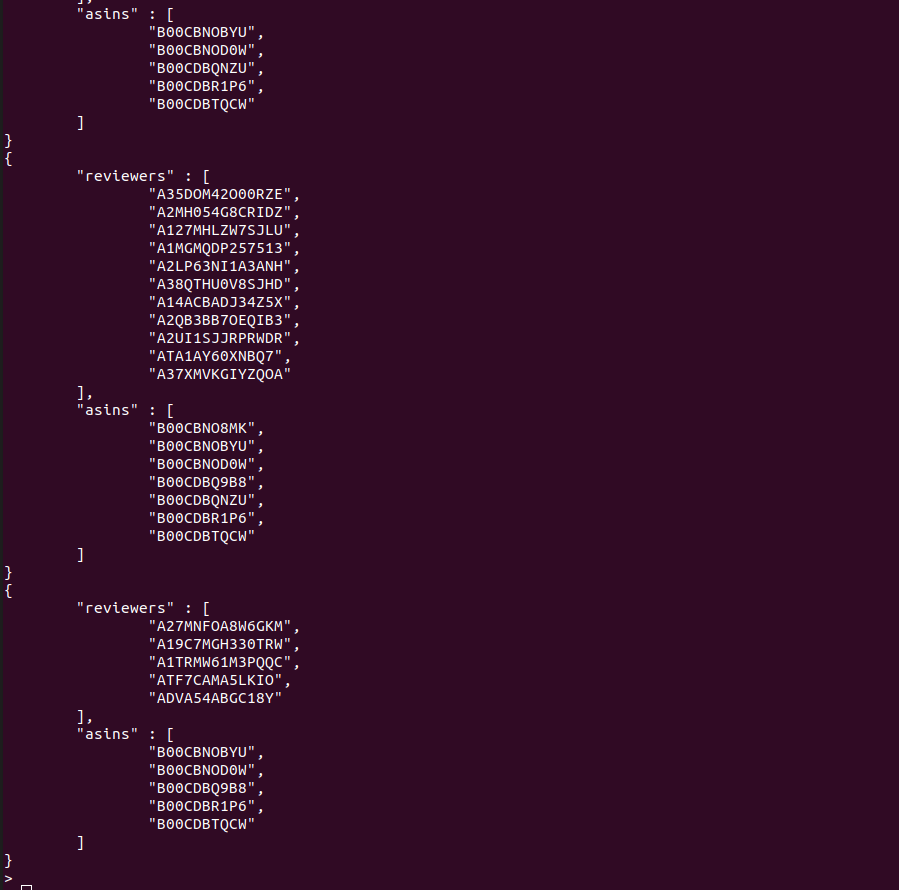
*Query:*

db.reviews.aggregate([{$group:{\_id: "$reviewerID", asins: {$push: "$asin"}}}, {$group:{\_id: "$asins", reviewers: {$push: "$\_id"}, reviewerCount: {$sum: 1}}}, {$sort: {reviewerCount: -1}}, {$project: {\_id: 0, asins: "$\_id", reviewers: 1}}, {$limit: 5}], {allowDiskUse: true}).pretty()

*Screenshot:*







#### Solution 2

*Wording:* We specify a list of product IDs/ASINs and check if there are any reviewers who reviewed all of the products in the specified list, no limitation for other product reviews

*Query:*

**In case of products: "B00CFM0P7Y", "B00EF3YF0Y", "B00EF45AHU"**

**Common reviewers: "A10TE5IGT94LMC", "A12JU025JDQ2UA", "A1DGD9DN29RMTI", "A2CQH041Y2RHD", "A2U9JB9JK34XZU", "A39HU59BYZ61ZB", "A3EP54ENFCRJ69", "A3W0TN7H2DTW7R", "AL5DKZ4E6XV45"**

db.reviews.aggregate([{$group: {\_id: "$asin", reviewers: {$push: "$reviewerID"}}}, {$match: {\_id: {$in: ["B00CFM0P7Y", "B00EF3YF0Y", "B00EF45AHU"]}}}, {$group: {\_id: 0, asins: {$push: "$\_id"}, reviewers: {$push: "$reviewers"}, initSet: {$first: "$reviewers"}}}, {$project: {\_id: 0, asins: 1, reviewersInCommon: {$reduce: {input: "$reviewers", initialValue: "$initSet", in: {$setIntersection: ["$$value", "$$this"]}}}}}]).pretty()

*Screenshot:*

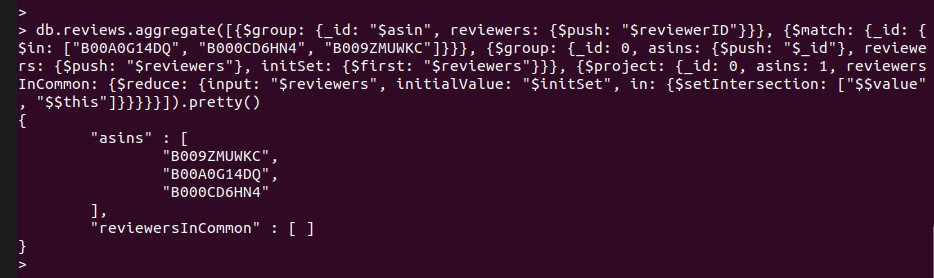


**In case of products: "B00A0G14DQ", "B000CD6HN4", "B009ZMUWKC"**

**Common reviewer: NONE**

db.reviews.aggregate([{$group: {\_id: "$asin", reviewers: {$push: "$reviewerID"}}}, {$match: {\_id: {$in: ["B00A0G14DQ", "B000CD6HN4", "B009ZMUWKC"]}}}, {$group: {\_id: 0, asins: {$push: "$\_id"}, reviewers: {$push: "$reviewers"}, initSet: {$first: "$reviewers"}}}, {$project: {\_id: 0, asins: 1, reviewersInCommon: {$reduce: {input: "$reviewers", initialValue: "$initSet", in: {$setIntersection: ["$$value", "$$this"]}}}}}]).pretty()

*Screenshot:*



*— Summary:*

Solution 1: Pretend we don’t know the specific group of products we need to group, our query will first group all products by reviewerID into a list called “asins”. Then, we will group all reviewer by “asins” into a list called “reviewers”. Snapshot shows asin list and reviewer list. This will give us combinations of products and reviewers who all reviewed those products.

Solution 2: Given a group of products by their ASIN, we can check if there are common reviewers. There are 2 examples provided, for a group of products with common reviewers and for a group with no common reviewers.

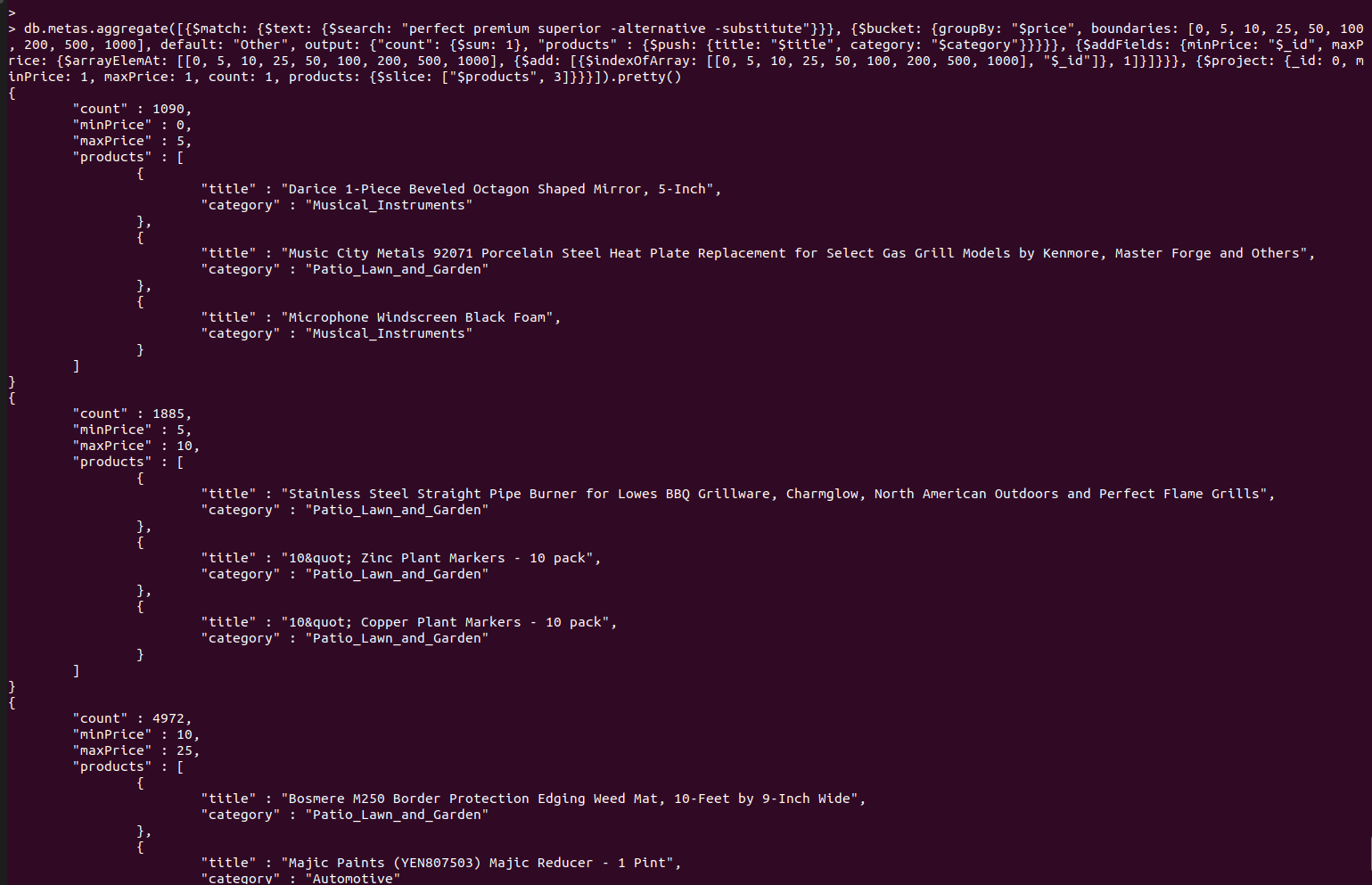
**G - Formulate an intriguing and complex query related to products text. Describe what the query is about, and its significance. Then provide the query statement as well as the output.**

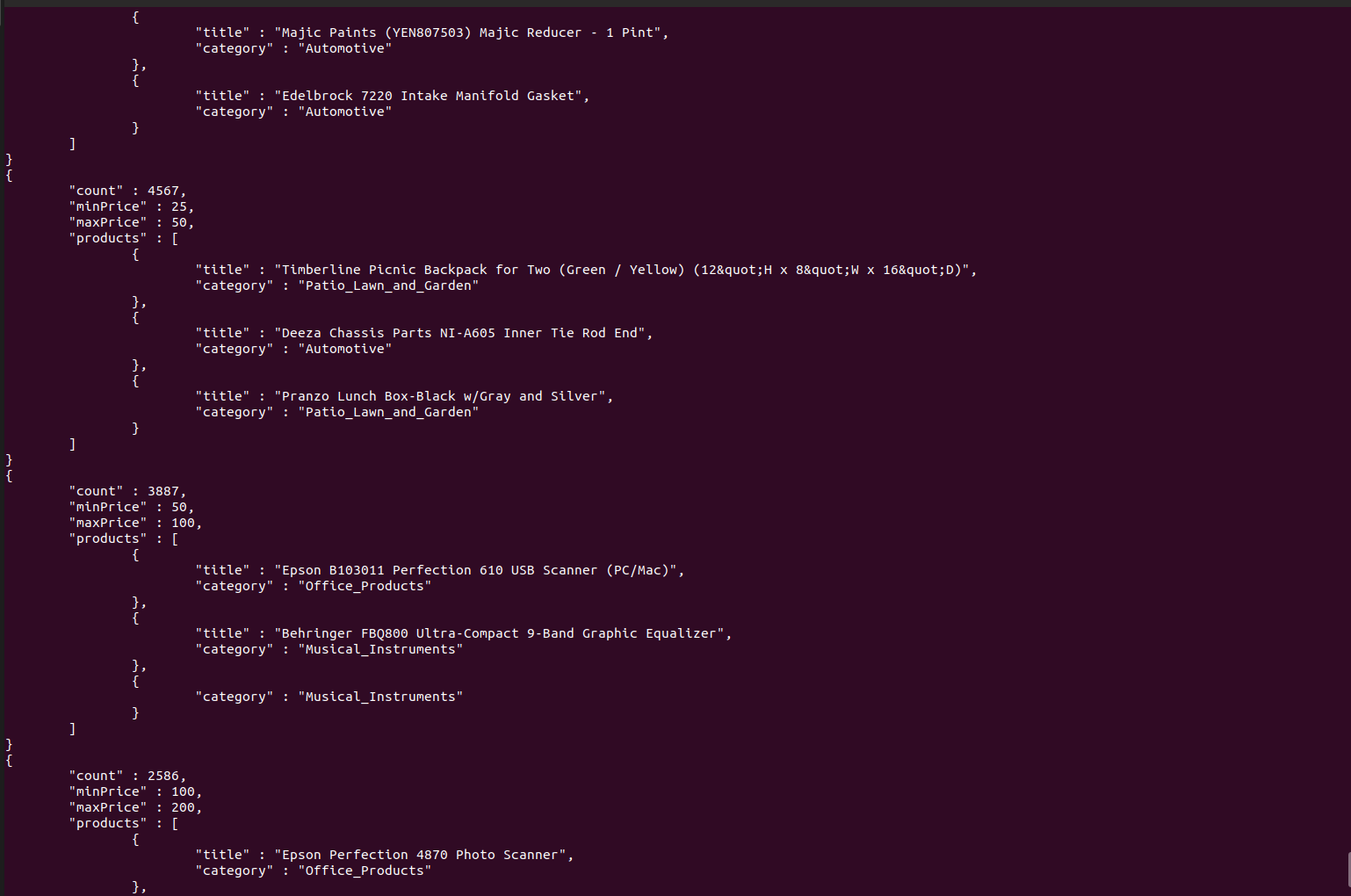
*Wording:* In the price ranges of 0-5, 5-10, 10-25, 25-50, 50-100, 100-200, 200-500, 500-1000 how many product’s description mentions the words “premium”, “perfect” or “superior” but not “alternative” or “substitute” (to get only OEM products), and show 3 of these products’ titles.

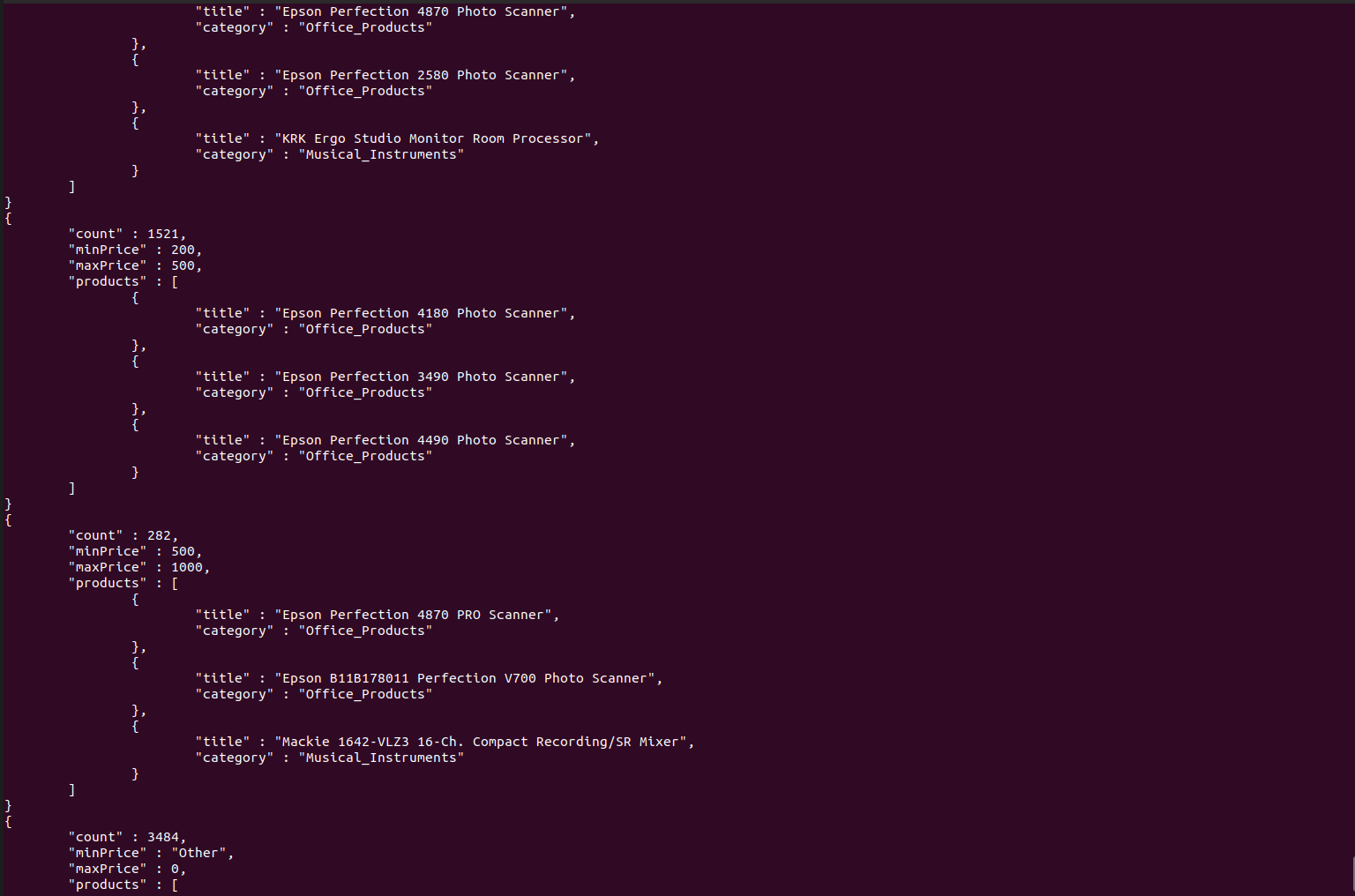
*Index:* db.metas.createIndex({description: "text"})

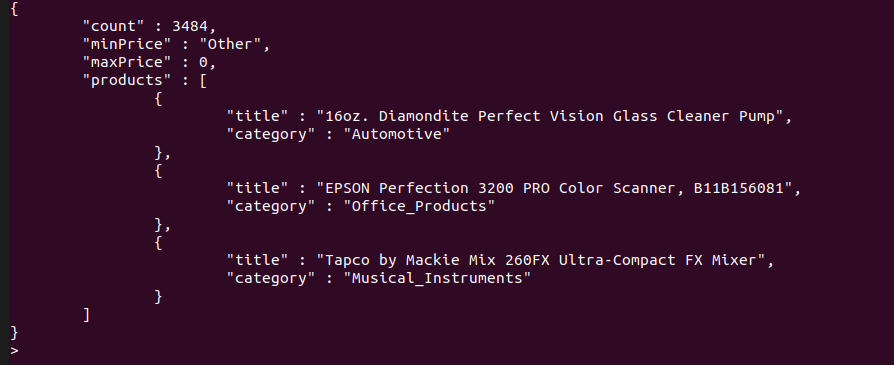
db.metas.aggregate([{$match: {$text: {$search: "perfect premium superior -alternative -substitute"}}}, {$bucket: {groupBy: "$price", boundaries: [0, 5, 10, 25, 50, 100, 200, 500, 1000], default: "Other", output: {"count": {$sum: 1}, "products" : {$push: {title: "$title", category: "$category"}}}}}, {$addFields: {minPrice: "$\_id", maxPrice: {$arrayElemAt: [[0, 5, 10, 25, 50, 100, 200, 500, 1000], {$add: [{$indexOfArray: [[0, 5, 10, 25, 50, 100, 200, 500, 1000], "$\_id"]}, 1]}]}}}, {$project: {\_id: 0, minPrice: 1, maxPrice: 1, count: 1, products: {$slice: ["$products", 3]}}}]).pretty()

*— Snapshot:*









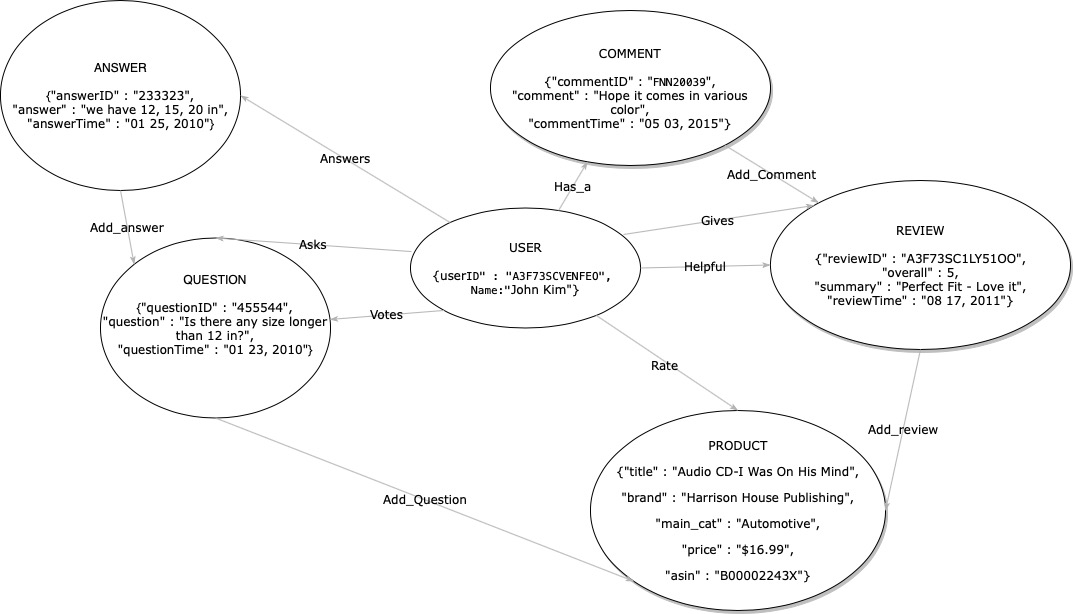
*— Summary:*

|  |  |
| --- | --- |
| Bin Price | Count |
| 0-5 | 1090 |
| 5-10 | 1885 |
| 10-25 | 4972 |
| 25-50 | 4567 |
| 50-100 | 3887 |
| 100-200 | 2586 |
| 200-500 | 1521 |
| 500-1000 | 282 |
| Other | 3484 |

Most of the products labeled as “premium” product have a price range between 10 and 100, but it seems like there can be “premium” products that cost only cents.

# Purpose 2

Graph:



Justification:

|  |  |
| --- | --- |
| **Nodes** | **Properties** |
| Product | ASIN, title, brand, main\_cat, price |
| Review | reviewID, summary, reviewTime |
| Question | questionID, question, questionTime |
| Answer | answerID, answer, answerTime |
| Comment | commentID, comment, commentTime |
| User | userID, Name |

|  |  |
| --- | --- |
| **Relationships** | **Properties** |
| Add\_answer | Answer - *relation* - Question |
| Add\_question | Question - *relation* - Product |
| Add\_comment | Comment - *relation* - Review |
| Add\_review | Review - *relation* - Product |
| Votes | User - *relation* - Question |
| Has\_a | Commenter - *relation* - Comment |
| Gives | User - *relation* - Review |
| Rate | User - *relation* - Product |
| Asks | User - *relation* - Question |
| Answers | User - *relation* - Answer |
| Helpful | User - relation - Review |

We made the system center around the user as the main point, with the highest betweenness and the lowest closeness, believing that most queries would involve the user in one way or another.

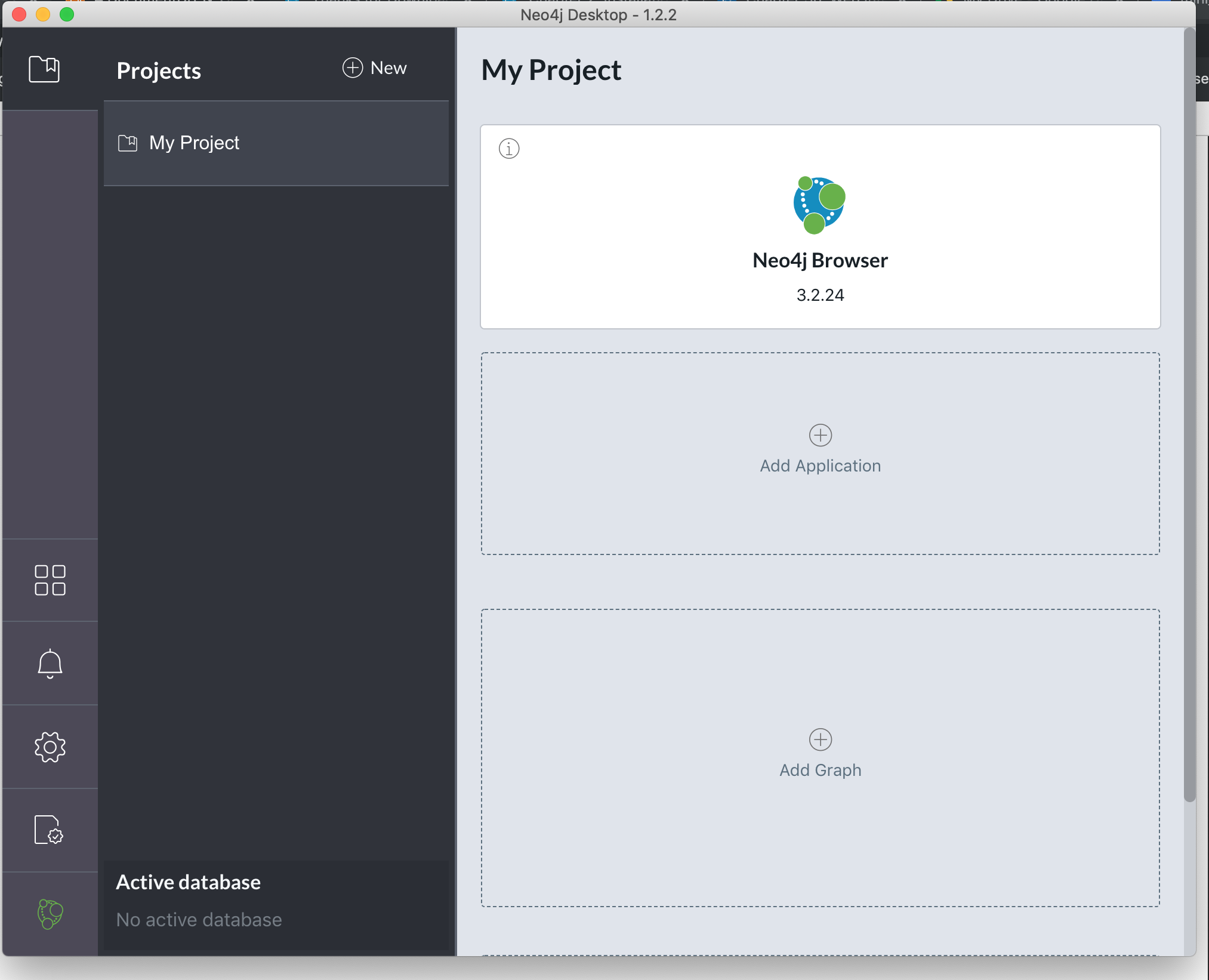
To avoid redundancy we did not make “reviewer” its own node, but instead a relationship between the “review” and the “user” nodes, keeping any information about the review stored in the review node.

We made a similar decision for the votes, as they are just expressed as a relationship between the user and the question they voted on.

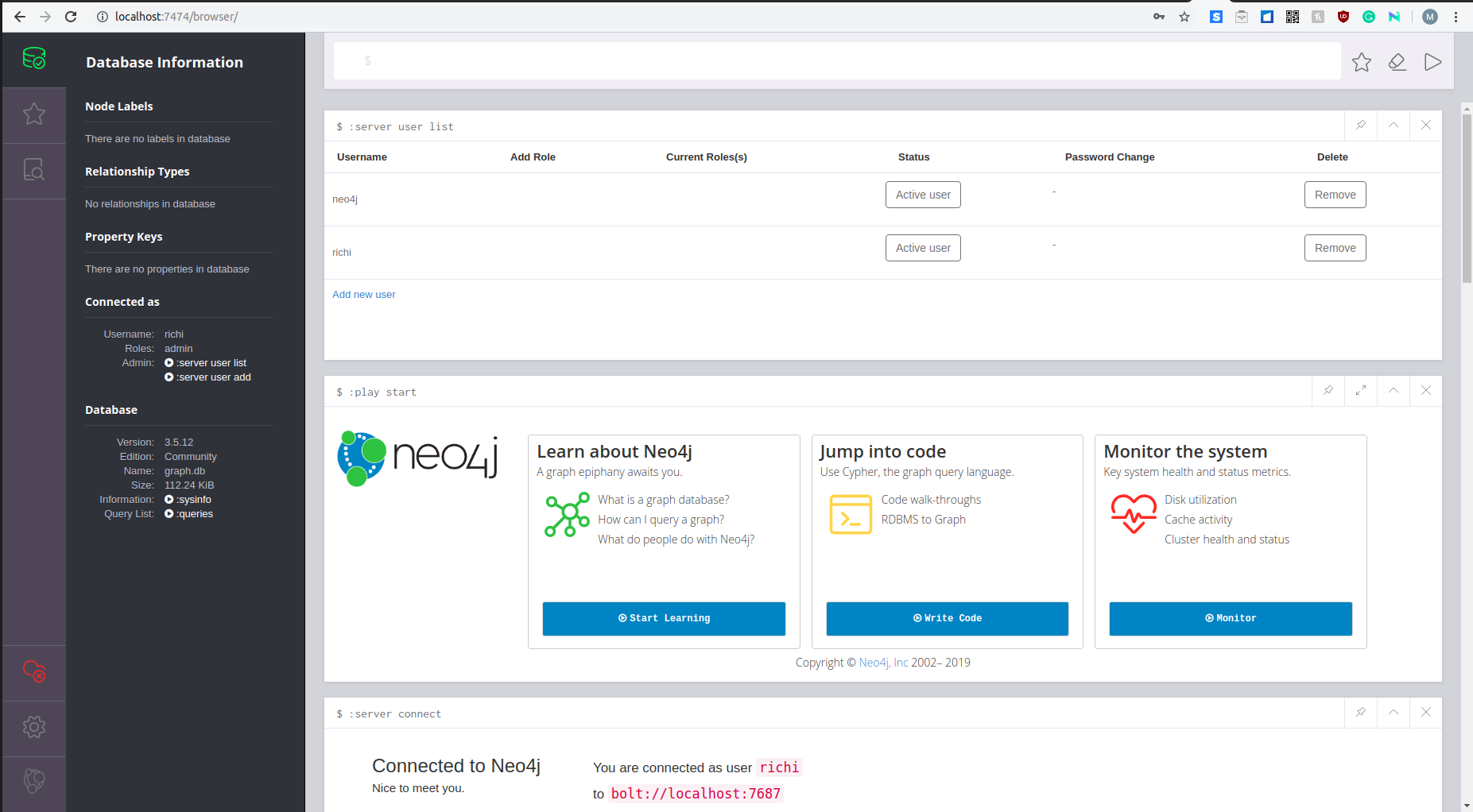
# Purpose 3

## Neo4j screenshots

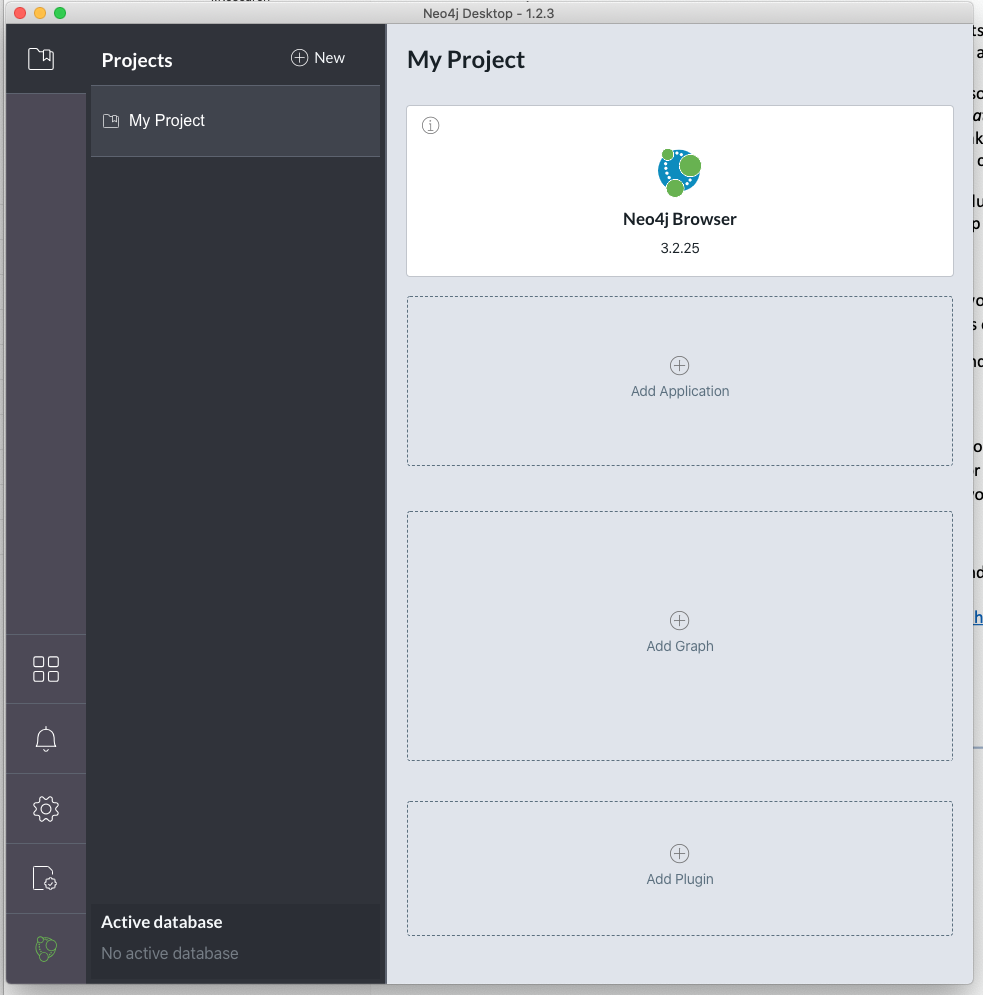
### Hao Nguyen



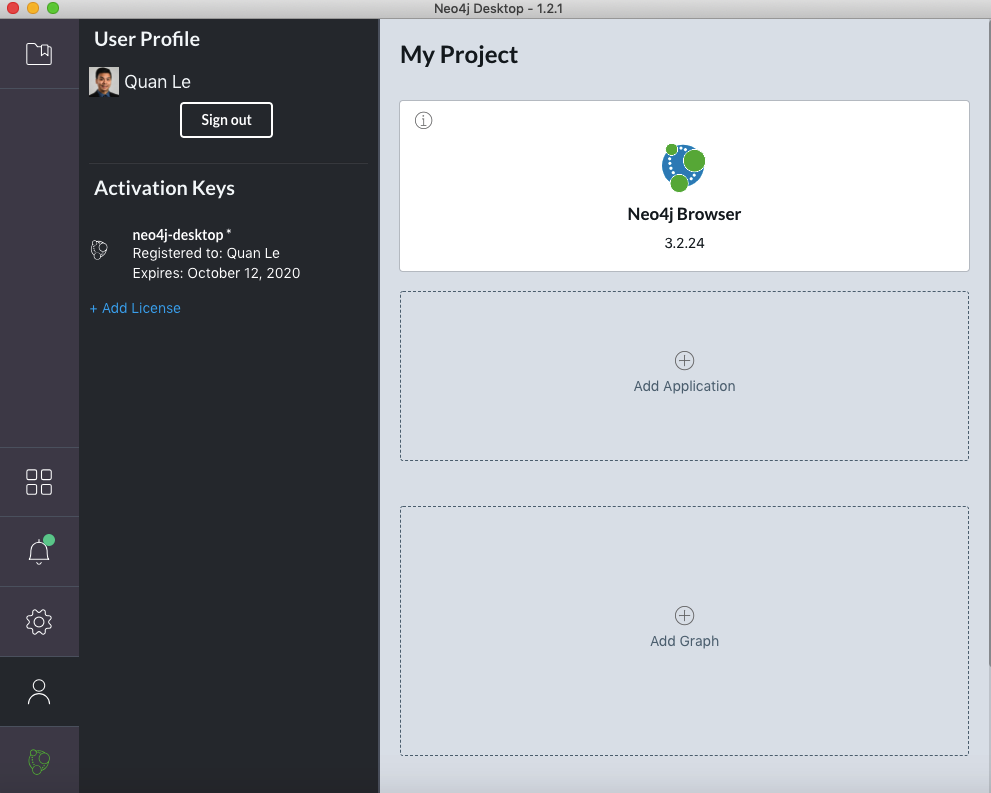
### Richard More



### Bob LeBow



### Quan Le



### Anirudh Chaudhary

