Term Project

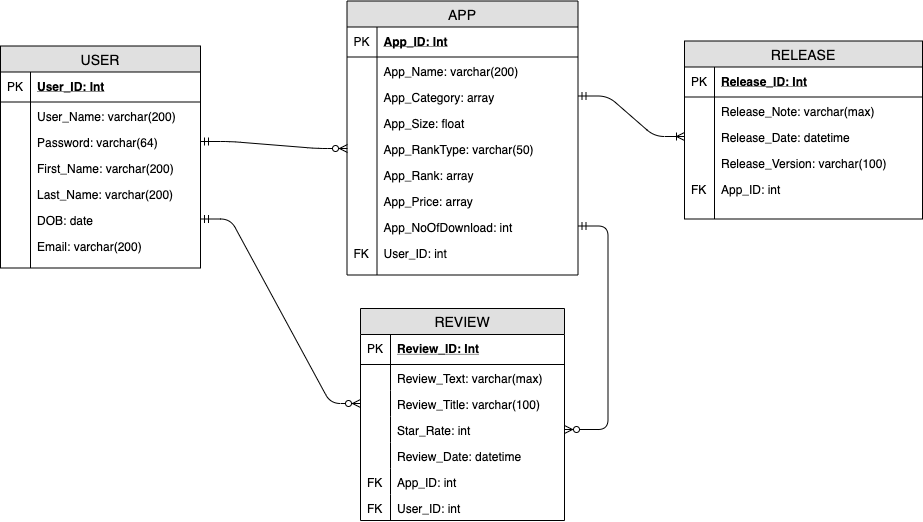
By: Anirudh Chaudhary, Quan Le, Bob LeBow, Richard More, Hao Nguyen

# A - Design

## MongoDB

**Design Pseudo Schema for MongoDB**

1. **Schema design**

****

1. **Pseudo Schema**

* **USER Collection**

{

"User\_ID": 1,

"User\_Name": “leQuan”,

"Password": "5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8",

"First\_Name": "Quan",

"Last\_Name": "Le",

"DBO": "11/22/1992",

"Email": "lequan@gmail.com"

}

* **APP Collection**

{

"App\_ID": 12,

"App\_Name": “Mathematics Quizzes with friends”,

"App\_category": [”entertainment”,”education”],

"App\_Size": 256000,

"App\_RankType": "Paid",

"App\_Rank": [{“Entertainment”: 64}, {“Education”: 15}],

"App\_Price": ["$1”,”$5”,”$6”],

"App\_NoOfDownload": 1500,

“User\_ID”: 1

}

* **Release Collection**

{

"Release\_ID": 33,

"Release\_Note": “Fix bug in matching opponent based on education”,

"Release\_Date": “11/22/2019 12:22:33.000”,

"Release\_Version": “2.4.1”,

"App\_ID": 12,

}

* **Review Collection**

{

"Review\_ID": 12,

"Review\_Text": “This is a really fun game, I can compete with my friend”,

"Review\_Tile": “Great App”,

"Star\_Rate": 5,

"Review\_Date": "12/1/2019 08:45:11.000",

"App\_ID": 12,

"User\_ID": 2

}

1. **Relationships**

* User CREATE App.
* App CONTAIN Release
* User CREATE Review
* App HAS Review

**Index of MongoDB**

* There will be a need for Text Index for Review\_Text field in REVIEW collection to assist identifying good or bad review
  + Code: db.Review.createIndex({Review\_Text:”text”})
* There will be a need of Text Index for Release\_Note field in Release collection to assist identifying which release is for fixing bugs, for modifying app features, or for other major changes
  + Code: db.Release.createIndex({Release\_Note:”text”})
* There need to be TLL apply to App\_Price to hold only 1 year of price history
  + Code:

{

"App\_id": 1,

"App\_category": [”entertainment”,”education”],

"App\_Size": 256000,

"App\_RankType": "Paid",

"App\_Rank": [{“Entertainment”: 64}, {“Education”: 15}],

"App\_Price": [

{“Price”:"$1”,”PriceChangedDate”: 623512},

{“Price”:"$5”,”PriceChangedDate”: 87345},

{“Price”:"$6”,”PriceChangedDate”: 1226}

],

"App\_NoOfDownload": 221000,

“User\_id”: 1

}

db.App.createIndex{{PriceChangedDate: 1}, expireAfterSeconds: 31556952}

**Constraints**

* **Email needs to be in format aaa@bbb.ccc**
  + **Code:**

db.createCollection( "User",

{ validator:

{Email: { $regex: /@/\./ } }

}

)

* **Star\_Rate needs to be between 1 and 5 stars**
  + **Code:**

db.createCollection( "Review",

{ validator:

{ Star\_Rate: { $in: [ 1, 2, 3, 4, 5 ] } }

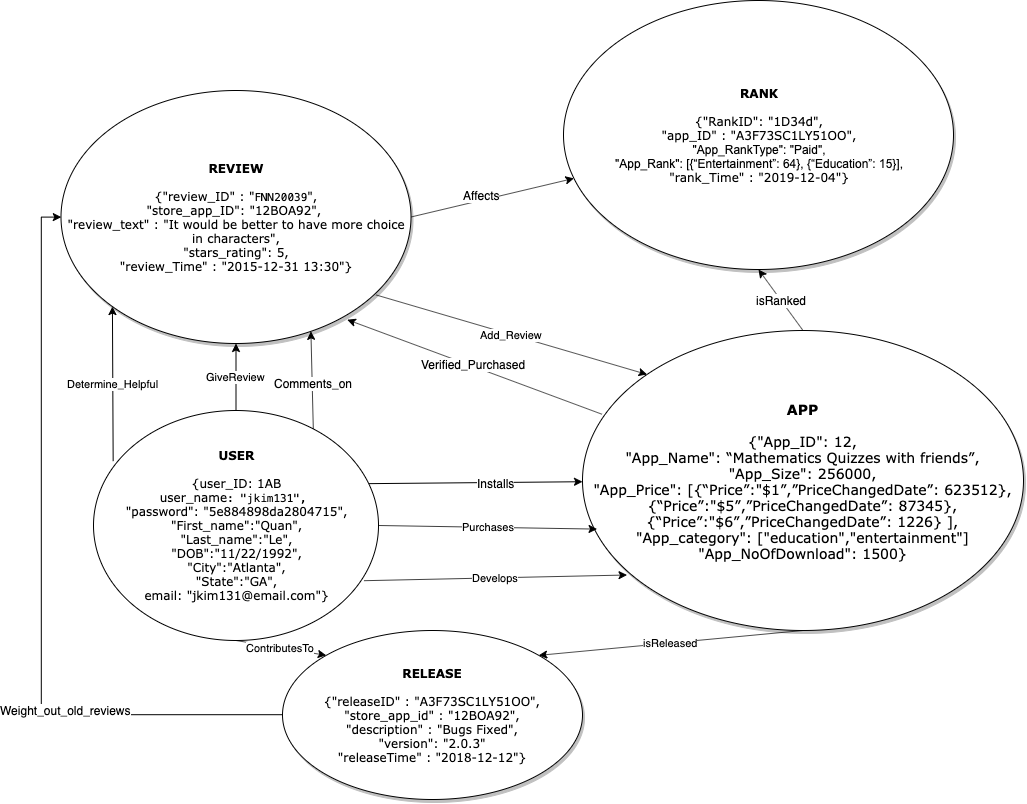
}

)

## Neo4j

**Design Pseudo Schema for Neo4j**

### Schema design

****

### Description

Node Properties

|  |  |
| --- | --- |
| Nodes | Properties |
| User | User\_ID, user\_name, password, First\_name, Last\_name, DOB, City, State, email |
| App | App\_ID, App\_name, App\_size, App\_Price, App\_category App\_NoOfDownload |
| Rank | RankID, App\_ID, App\_Ranktype, rank\_time |
| Release | releaseID, store\_app\_id, description, version, releaseTime |
| Review | reviewID, store\_app\_ID, review\_text, starts\_rating, review\_Time |

Relationship Properties

|  |  |
| --- | --- |
| Relationships | Properties |
| Installs | User - relation - App |
| Purchases | User - relation - App |
| Develops | User - relation - App |
| isReleased | App - relation - Release |
| ContributesTo | User - relation - Release |
| GiveReview | User - relation - Review |
| Comments\_on | User - relation - Review |
| Determine\_Helpful | User - relation - Review |
| Weight\_out\_old\_reviews | Release - relation - Review |
| Add\_Review | Review - relation - App |
| Verified\_Purchased | App - relation - Review |
| Affects | Review - relation - Rank |
| isRanked | App - relation - Rank |

### 

### Justification

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. Since users can be developers or customers, they significantly affect the node App.

To avoid redundancy, we made “rank” its own node because an app can have multiple categories, and in each category, they have their own ranking system along with overall ranking and rank type’s ranking.

In our Mini-project 1, we researched the weight of the reviews. For every new release, the analytic tool will determine if the review would be biased or still relevant to the new version of the app. Therefore, we add a relationship called “Weight\_out\_old\_review”. In addition, there is a relationship called “affects” between review and ranking as we believe the amount of stars rating and reviews will affect the ranking of the app.

To validate the review, there is a relationship called “Verified\_Purchased” between App and reviews that acts as a verifier.

A user can add a review post to the app, and another user who is a developer can comment on the review. Other users can then decide if the comment or review itself is helpful or unhelpful.

# B - Basic understanding of the data - MongoDB

### Import commands and Richi magic

sed -i -e "s/\\\|//g" -e "s/\"//g" -e "s/|/\"/g" -e "1 i\id,store\_app\_id,app\_name,rank\_type,category,rank,date" /var/opt/app\_rankings.txt

sed -i -e "s/\\\|//g" -e "s/\"//g" -e "s/|/\"/g" /var/opt/app\_releasenotes.txt

sed -i -e "s/\"//g" -e "s/|,|/\",\"/g" -e "s/^|/\"/g" -e "s/|$/\"/g" -e "1 i\id,store\_app\_id,review\_id,title,body,star\_rating,date,username,user\_apple\_id,user\_reviews\_url" /var/opt/app\_reviews.txt

mongoimport --host localhost:27017 -d 'apple' -c 'rankings' --type csv --headerline --file /var/opt/app\_rankings.txt

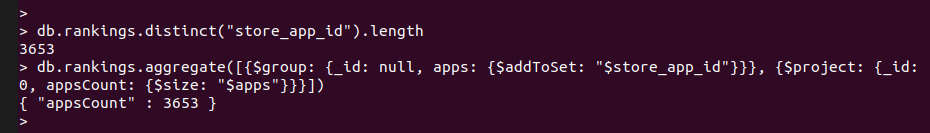
mongoimport --host localhost:27017 -d 'apple' -c 'releases' --type csv --headerline --file /var/opt/app\_releasenotes.txt

mongoimport --host localhost:27017 -d 'apple' -c 'reviews' --type csv --headerline --file /var/opt/app\_reviews.txt

### What is the overall number of apps?

db.rankings.distinct("store\_app\_id").length

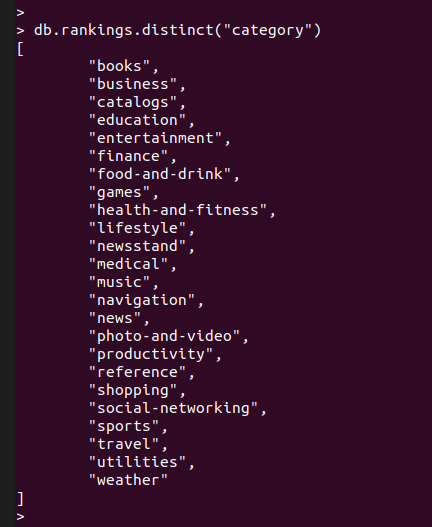
db.rankings.aggregate([{$group: {\_id: null, apps: {$addToSet: "$store\_app\_id"}}}, {$project: {\_id: 0, appsCount: {$size: "$apps"}}}])



Summary: There are 3653 apps in the dataset.

### What are the different categories of apps?

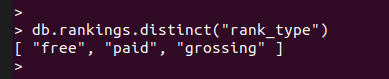
db.rankings.distinct("category")



Summary: There are a wide range of categories of apps represented in the dataset.

### What ranking types are the apps listed in?

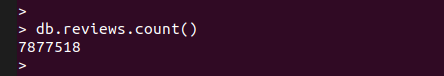
db.rankings.distinct("rank\_type")



Summary: The three ranking types are free, paid, and grossing.

### What is the overall number of reviews?

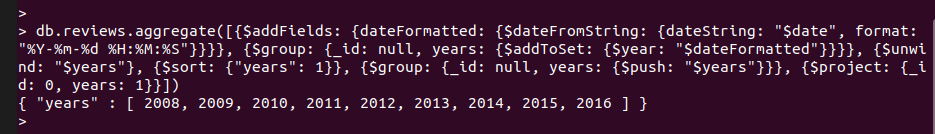
db.reviews.count()



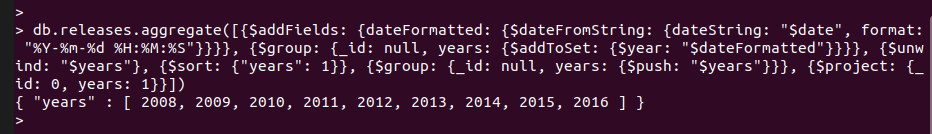
Summary: There are 7877518 reviews in the reviews collections.

### Reviews and releases of which years are covered?

db.reviews.aggregate([{$addFields: {dateFormatted: {$dateFromString: {dateString: "$date", format: "%Y-%m-%d %H:%M:%S"}}}}, {$group: {\_id: null, years: {$addToSet: {$year: "$dateFormatted"}}}}, {$unwind: "$years"}, {$sort: {"years": 1}}, {$group: {\_id: null, years: {$push: "$years"}}}, {$project: {\_id: 0, years: 1}}])



db.releases.aggregate([{$addFields: {dateFormatted: {$dateFromString: {dateString: "$date", format: "%Y-%m-%d %H:%M:%S"}}}}, {$group: {\_id: null, years: {$addToSet: {$year: "$dateFormatted"}}}}, {$unwind: "$years"}, {$sort: {"years": 1}}, {$group: {\_id: null, years: {$push: "$years"}}}, {$project: {\_id: 0, years: 1}}])



Summary: We have app releases and reviews from 2008 till 2016.

### What is the overall number of reviewers?

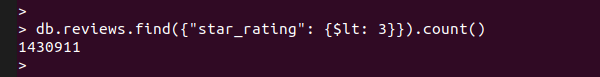
db.reviews.aggregate([{$group: {\_id: null, reviewers: {$addToSet: "$user\_apple\_id"}}}, {$project: {\_id: 0, reviewerCount: {$size: "$reviewers"}}}])



Summary: There are 5777950 reviewers in the dataset.

### What is the overall number of reviews with ratings less than 3?

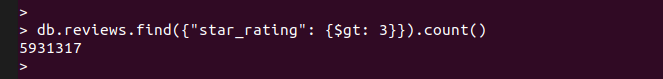
db.reviews.find({"star\_rating": {$lt: 3}}).count()



Summary: There are 1430911 reviews with a rating of 1 or 2.

### What is the overall number of reviews with ratings more than 3?

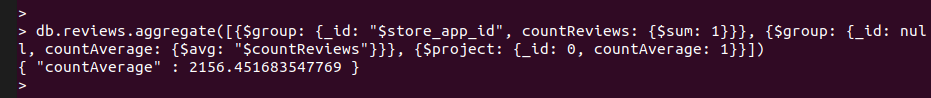
db.reviews.find({"star\_rating": {$gt: 3}}).count()



Summary: There are 5931317 reviews with a rating of 4 or 5. Looking at the result of the previous question, people seem to review apps more that they like rather than review the ones they do not. This might mean that users either just don’t use “bad” apps or they do not want to report their problems, in this case they do not care for the improvement of the app as much.

### What is the average number of reviews per app?

db.reviews.aggregate([{$group: {\_id: "$store\_app\_id", countReviews: {$sum: 1}}}, {$group: {\_id: null, countAverage: {$avg: "$countReviews"}}}, {$project: {\_id: 0, countAverage: 1}}])

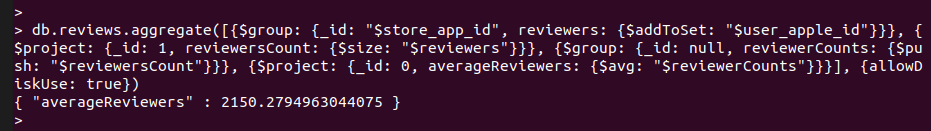


Summary: On average there are 2156 reviews for every app

### What is the average number of reviewers for the apps?

Note: Not per app, but overall

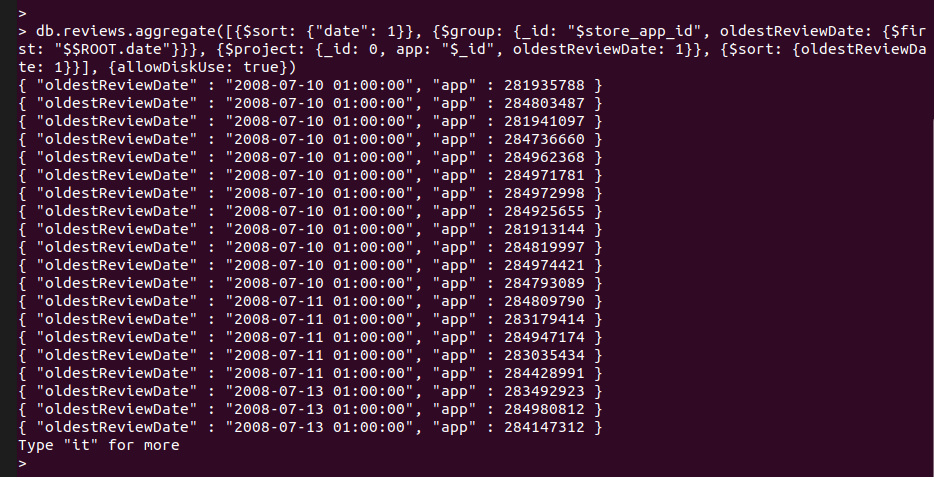
db.reviews.aggregate([{$group: {\_id: "$store\_app\_id", reviewers: {$addToSet: "$user\_apple\_id"}}}, {$project: {\_id: 1, reviewersCount: {$size: "$reviewers"}}}, {$group: {\_id: null, averageReviewers: {$avg: "$reviewersCount"}}}, {$project: {\_id: 0, averageReviewers: 1}}], {allowDiskUse: true})



Summary: On average there are 2150 unique reviewers for every app.

### What is the date of the first review per app?

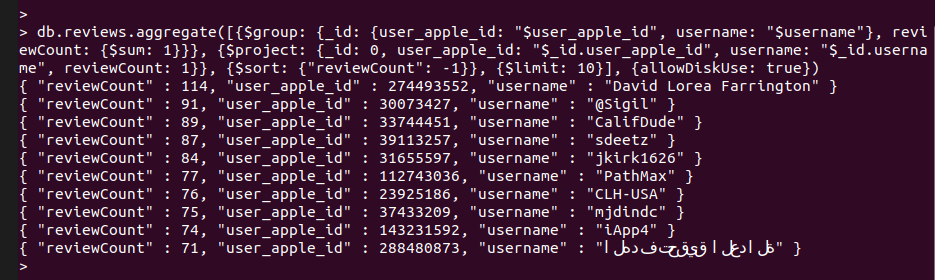
db.reviews.aggregate([{$sort: {"date": 1}}, {$group: {\_id: "$store\_app\_id", oldestReviewDate: {$first: "$$ROOT.date"}}}, {$project: {\_id: 0, app: "$\_id", oldestReviewDate: 1}}, {$sort: {oldestReviewDate: 1}}], {allowDiskUse: true})



Summary: Above in the screenshot we can see the date of the first recorded review’s date for some applications.

### Who are the top 10 most prolific reviewers?

db.reviews.aggregate([{$group: {\_id: {user\_apple\_id: "$user\_apple\_id", username: "$username"}, reviewCount: {$sum: 1}}}, {$project: {\_id: 0, user\_apple\_id: "$\_id.user\_apple\_id", username: "$\_id.username", reviewCount: 1}}, {$sort: {"reviewCount": -1}}, {$limit: 10}], {allowDiskUse: true})

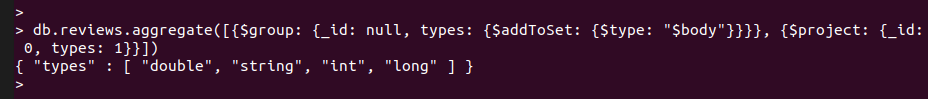


Summary: Above in the screenshot we can find the top 10 reviewers based on the number of reviews they have written. The Number 1 is David Lorea Farrington with 114 reviews.

### Who are the top 10 most verbose reviewers?

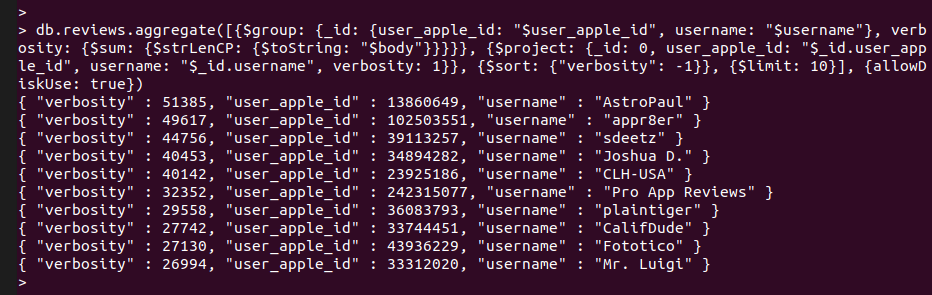
Since the strLenCP takes only string expressions we have to make sure that the body field has only strings.

db.reviews.aggregate([{$group: {\_id: null, types: {$addToSet: {$type: "$body"}}}}, {$project: {\_id: 0, types: 1}}])



As it turns out the types vary, this way for using the strLenCP we have to convert the contents of that field to string first.

db.reviews.aggregate([{$group: {\_id: {user\_apple\_id: "$user\_apple\_id", username: "$username"}, verbosity: {$sum: {$strLenCP: {$toString: "$body"}}}}}, {$project: {\_id: 0, user\_apple\_id: "$\_id.user\_apple\_id", username: "$\_id.username", verbosity: 1}}, {$sort: {"verbosity": -1}}, {$limit: 10}], {allowDiskUse: true})

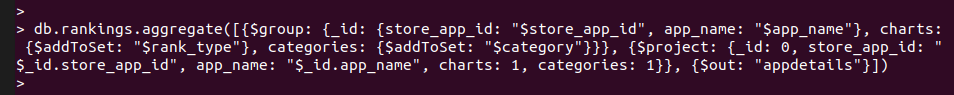


Summary: The above screenshot shows the top 10 reviewers based on the length of their reviews. The Number 1 is AstroPaul with 51385 characters written overall in his/her reviews.

### How many reviews are there per chart type?

For this query we will need to store the details of an app in a collection, since the data available in the rankings collection is not compressed enough.

db.rankings.aggregate([{$group: {\_id: {store\_app\_id: "$store\_app\_id", app\_name: "$app\_name"}, charts: {$addToSet: "$rank\_type"}, categories: {$addToSet: "$category"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", app\_name: "$\_id.app\_name", charts: 1, categories: 1}}, {$out: "appdetails"}])



With this query we will have a collection named appdetails that contain for every app its: id, name, chart types, and categories it is in.

To get the number of reviews per chart type:

db.reviews.aggregate([{$lookup: {from: "appdetails", localField: "store\_app\_id", foreignField: "store\_app\_id", as: "appdetail"}}, {$replaceRoot: {newRoot: {$mergeObjects: [{$arrayElemAt: ["$appdetail", 0]}, "$$ROOT"]}}}, {$project: {\_id: 0, store\_app\_id: 1, charts: 1}}, {$unwind: "$charts"}, {$group: {\_id: "$charts", count: {$sum: 1}}}, {$project: {\_id: 0, chart: "$\_id", count: 1}}], {allowDiskUse: true}).pretty()

Summary: Unfortunately this query takes forever to complete, so we do not have results. This process maxed out a core, but even after 4 hours no result was produced.

### How many reviews are there per category?

Using the above mentioned appdetails collection

db.reviews.aggregate([{$lookup: {from: "appdetails", localField: "store\_app\_id", foreignField: "store\_app\_id", as: "appdetail"}}, {$replaceRoot: {newRoot: {$mergeObjects: [{$arrayElemAt: ["$appdetail", 0]}, "$$ROOT"]}}}, {$project: {\_id: 0, store\_app\_id: 1, categories: 1}}, {$unwind: "$categories"}, {$group: {\_id: "$categories", count: {$sum: 1}}}, {$project: {\_id: 0, category: "$\_id", count: 1}}], {allowDiskUse: true}).pretty()

Summary: Unfortunately this query takes forever to complete, so we do not have results. Same hardware issues as above.

# C - Analytics

## MongoDB

### (MongoDB) Propose a list of no less than 5 metrics (such as review histogram, list of top 10 most recent reviews, list of top 10 most prolific reviewers...). Justify your choice and implement the needed queries.

Note: In this section, we will provide the base query and a query and snapshot for the app { "store\_app\_id" : 318698524, "app\_name" : "Google Voice" }

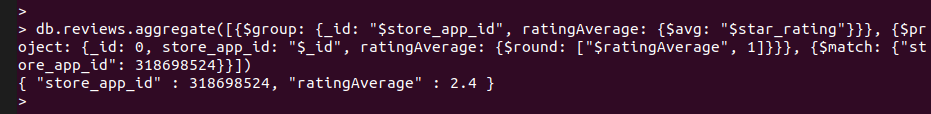
#### Average star rating

db.reviews.aggregate([{$group: {\_id: "$store\_app\_id", ratingAverage: {$avg: "$star\_rating"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", ratingAverage: {$round: ["$ratingAverage", 1]}}}])



For Google Voice:

db.reviews.aggregate([{$group: {\_id: "$store\_app\_id", ratingAverage: {$avg: "$star\_rating"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", ratingAverage: {$round: ["$ratingAverage", 1]}}}, {$match: {"store\_app\_id": 318698524}}])

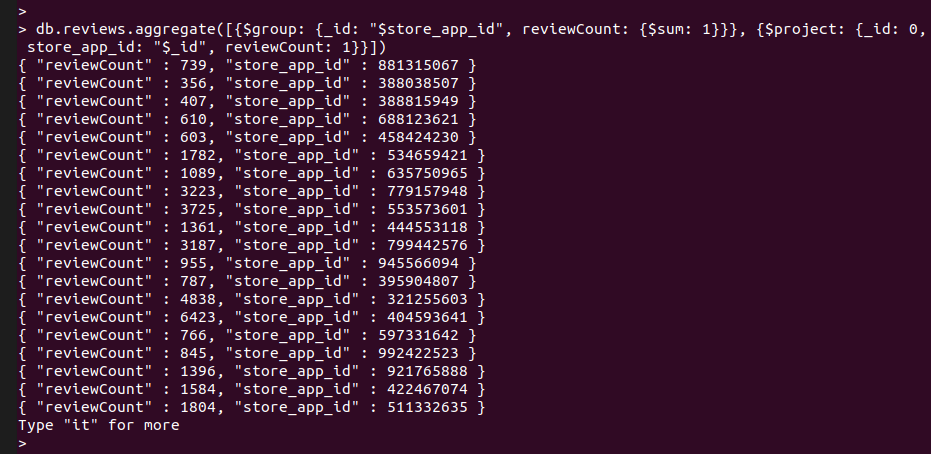


Summary: For Google Voice the average rating is 2.4 stars.

Justification: Rating average is a widely used metric for applications, products (like on Amazon, Aliexpress, etc), so it is a must have for system where reviews and ratings are collected. With this the user can easily choose between two apps having similar functionalities but different ratings - hoping that the reviews are not fake.

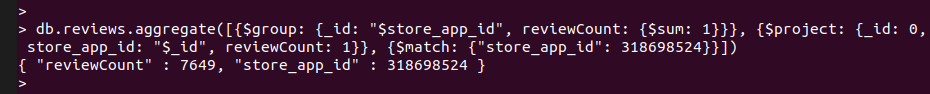
#### Number of reviews/ratings received for the app

db.reviews.aggregate([{$group: {\_id: "$store\_app\_id", reviewCount: {$sum: 1}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", reviewCount: 1}}])



For Google Voice:

db.reviews.aggregate([{$group: {\_id: "$store\_app\_id", reviewCount: {$sum: 1}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", reviewCount: 1}}, {$match: {"store\_app\_id": 318698524}}])



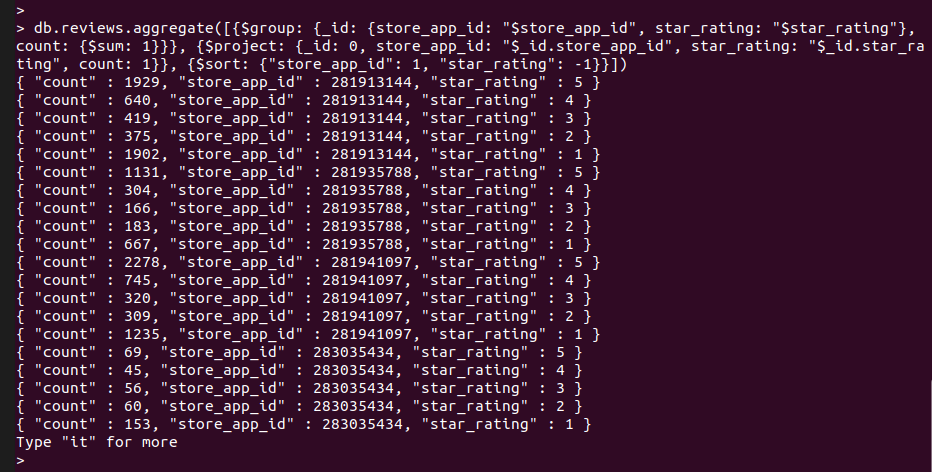
Summary: For Google Voice there has been 7649 reviews received since it has been published first.

Justification: It is important to know how many users have reviewed the app, since just on its own a rating or 2.4 out of 5 does not mean anything if it is the average rating of 5 users, but if it is the rating coming from 1000 people we can accept the rating as the true value of the app. Just again with this one, it is a widely used metric for apps, products, services, etc.

#### Review star rating histogram as seen on Amazon

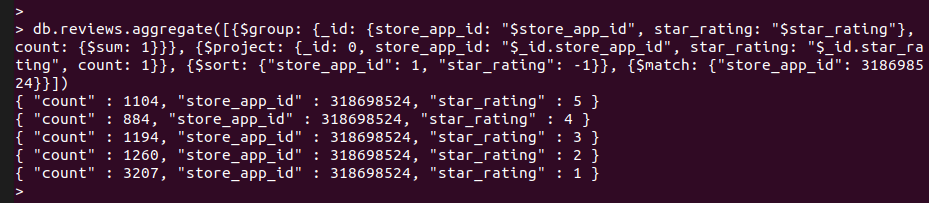
Shorter solution with counts in separate documents:

db.reviews.aggregate([{$group: {\_id: {store\_app\_id: "$store\_app\_id", star\_rating: "$star\_rating"}, count: {$sum: 1}}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", star\_rating: "$\_id.star\_rating", count: 1}}, {$sort: {"store\_app\_id": 1, "star\_rating": -1}}])



For Google Voice:

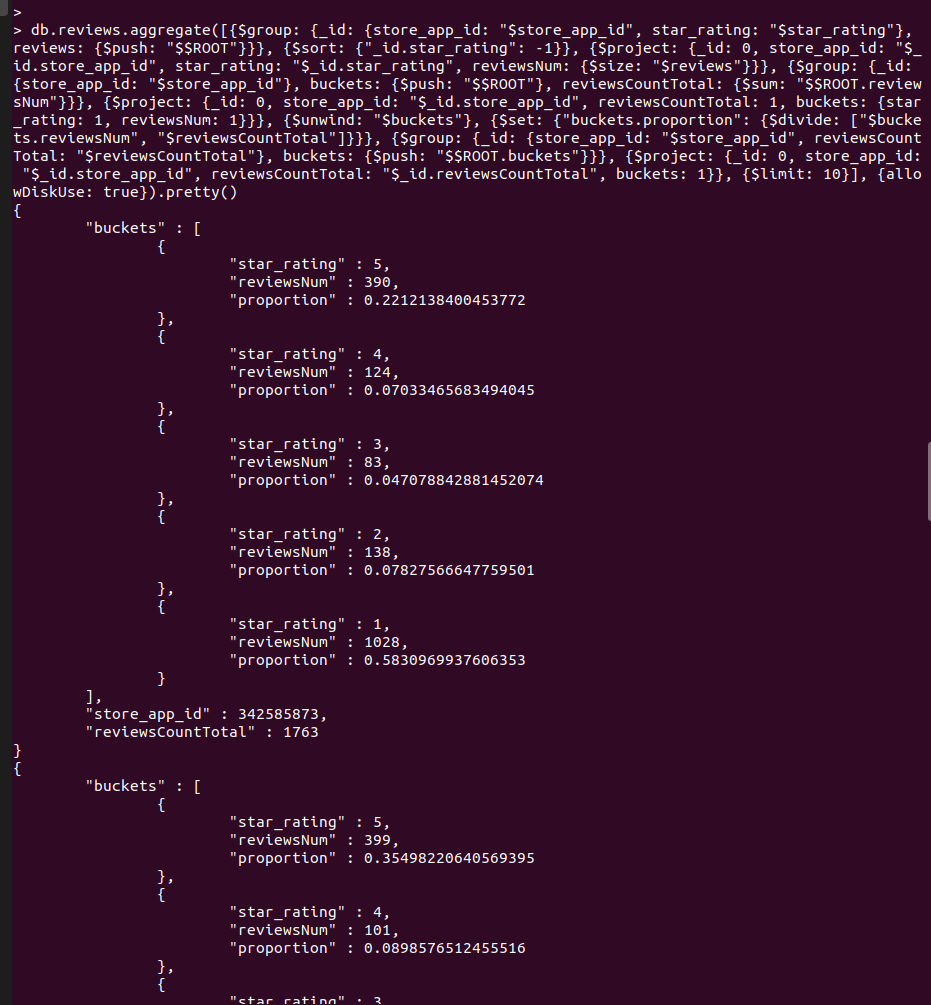
db.reviews.aggregate([{$group: {\_id: {store\_app\_id: "$store\_app\_id", star\_rating: "$star\_rating"}, count: {$sum: 1}}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", star\_rating: "$\_id.star\_rating", count: 1}}, {$sort: {"store\_app\_id": 1, "star\_rating": -1}}, {$match: {"store\_app\_id": 318698524}}])

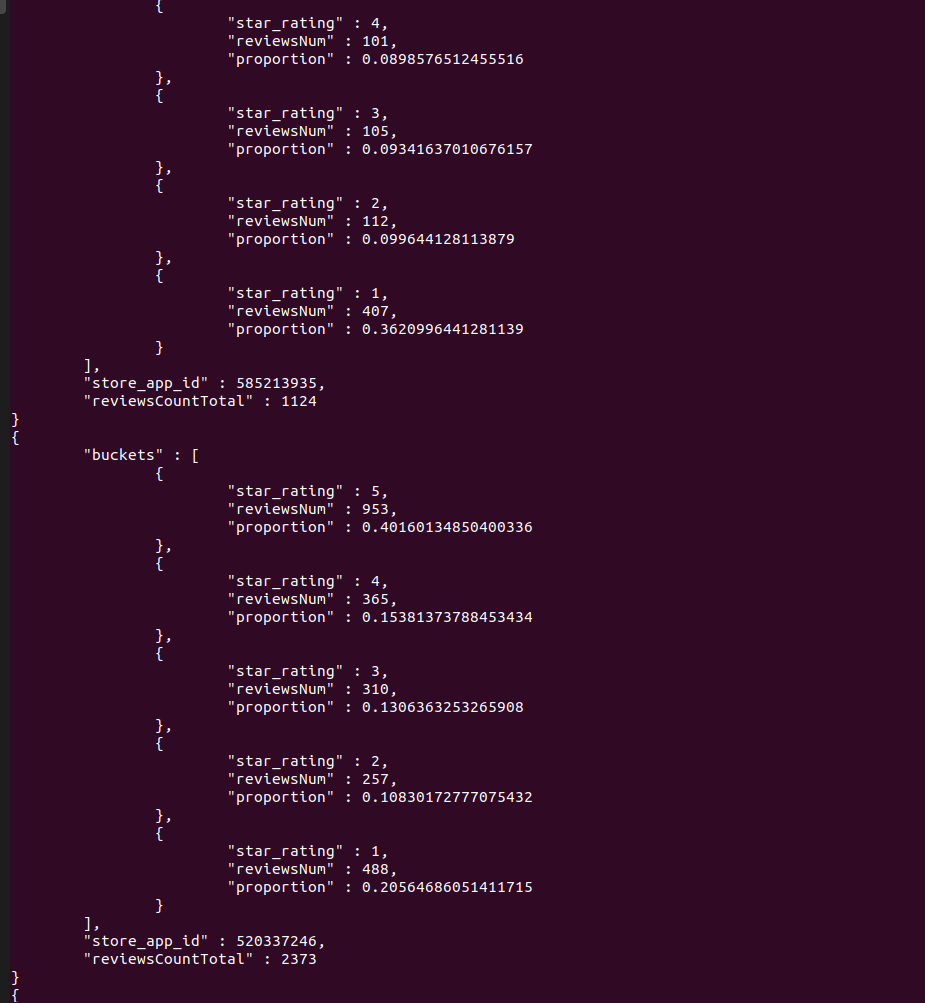


This solution is easier to read, but for every app, there are several (maximum 5) documents that have to be read to gather the data for a histogram, and for a “progress bar-like” visualization the percentages have to be calculated after getting the data.

Longer solution with counts and proportions in one document:

db.reviews.aggregate([{$group: {\_id: {store\_app\_id: "$store\_app\_id", star\_rating: "$star\_rating"}, reviews: {$push: "$$ROOT"}}}, {$sort: {"\_id.star\_rating": -1}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", star\_rating: "$\_id.star\_rating", reviewsNum: {$size: "$reviews"}}}, {$group: {\_id: {store\_app\_id: "$store\_app\_id"}, buckets: {$push: "$$ROOT"}, reviewsCountTotal: {$sum: "$$ROOT.reviewsNum"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", reviewsCountTotal: 1, buckets: {star\_rating: 1, reviewsNum: 1}}}, {$unwind: "$buckets"}, {$set: {"buckets.proportion": {$divide: ["$buckets.reviewsNum", "$reviewsCountTotal"]}}}, {$group: {\_id: {store\_app\_id: "$store\_app\_id", reviewsCountTotal: "$reviewsCountTotal"}, buckets: {$push: "$$ROOT.buckets"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", reviewsCountTotal: "$\_id.reviewsCountTotal", buckets: 1}}, {$limit: 10}], {allowDiskUse: true}).pretty()





For Google Voice:

db.reviews.aggregate([{$group: {\_id: {store\_app\_id: "$store\_app\_id", star\_rating: "$star\_rating"}, reviews: {$push: "$$ROOT"}}}, {$sort: {"\_id.star\_rating": -1}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", star\_rating: "$\_id.star\_rating", reviewsNum: {$size: "$reviews"}}}, {$group: {\_id: {store\_app\_id: "$store\_app\_id"}, buckets: {$push: "$$ROOT"}, reviewsCountTotal: {$sum: "$$ROOT.reviewsNum"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", reviewsCountTotal: 1, buckets: {star\_rating: 1, reviewsNum: 1}}}, {$unwind: "$buckets"}, {$set: {"buckets.proportion": {$divide: ["$buckets.reviewsNum", "$reviewsCountTotal"]}}}, {$group: {\_id: {store\_app\_id: "$store\_app\_id", reviewsCountTotal: "$reviewsCountTotal"}, buckets: {$push: "$$ROOT.buckets"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", reviewsCountTotal: "$\_id.reviewsCountTotal", buckets: 1}}, {$match: {"store\_app\_id": 318698524}}, {$limit: 10}], {allowDiskUse: true}).pretty()



This solution is harder to read, involves more operations, but the final result is more compact and contains more information: all the histogram data fields can be found in the subdocuments in the buckets: star rating value, count, and the proportion which is what we would need if we want to visualize the data in a different format.

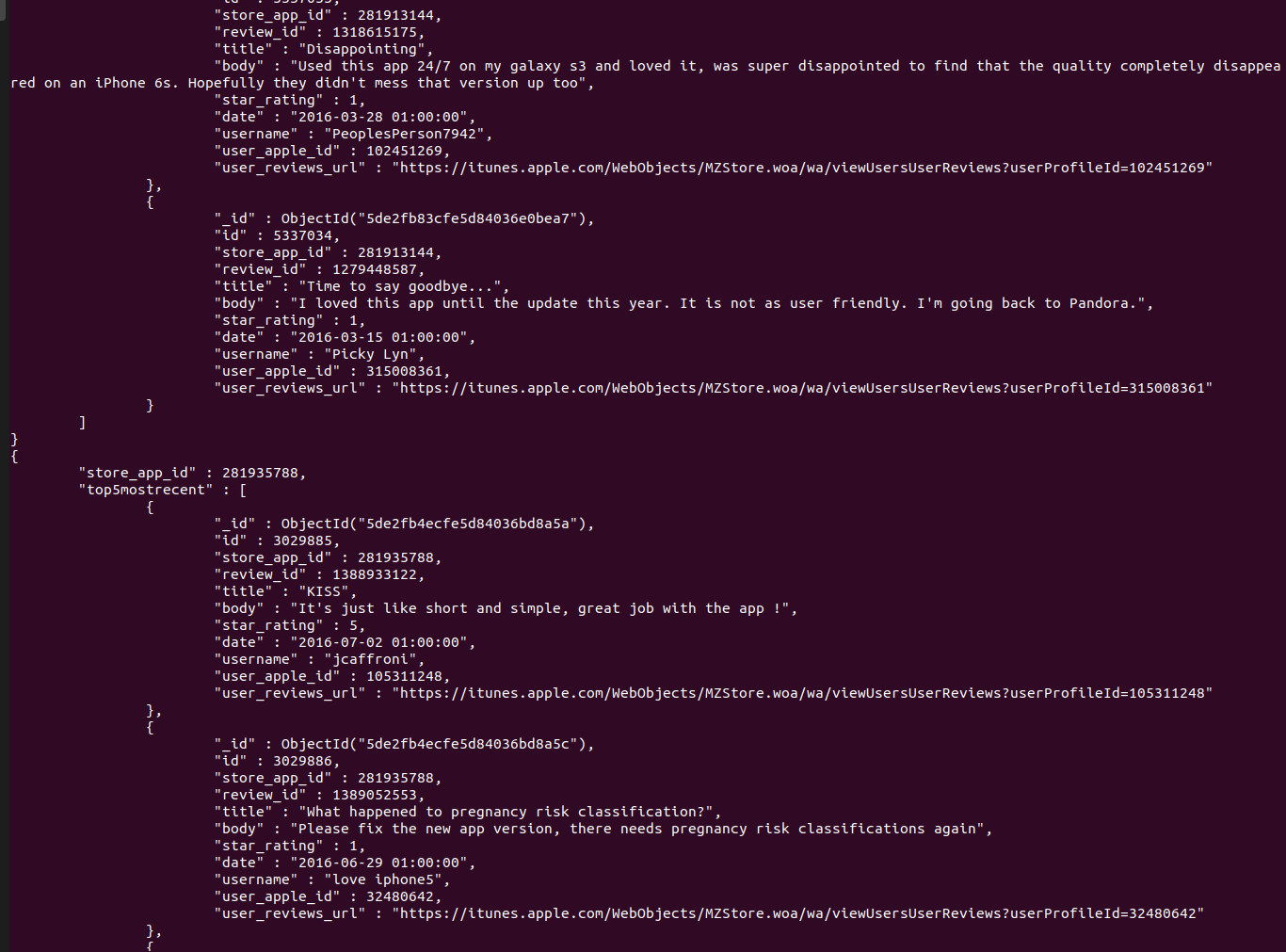
Summary: For Google Voice there are 1104 5-star reviews which is 14% of all its reviews, 884 4-star reviews which is 11% of all its reviews, 1194 3-star reviews which is 15% of all its reviews, 1260 2-star reviews which is 16% of its reviews, and 3207 1-star reviews which take up 41% of all its reviews. The displayed proportions are really helpful to immediately give an overview of the rating histogram even without a visual display.

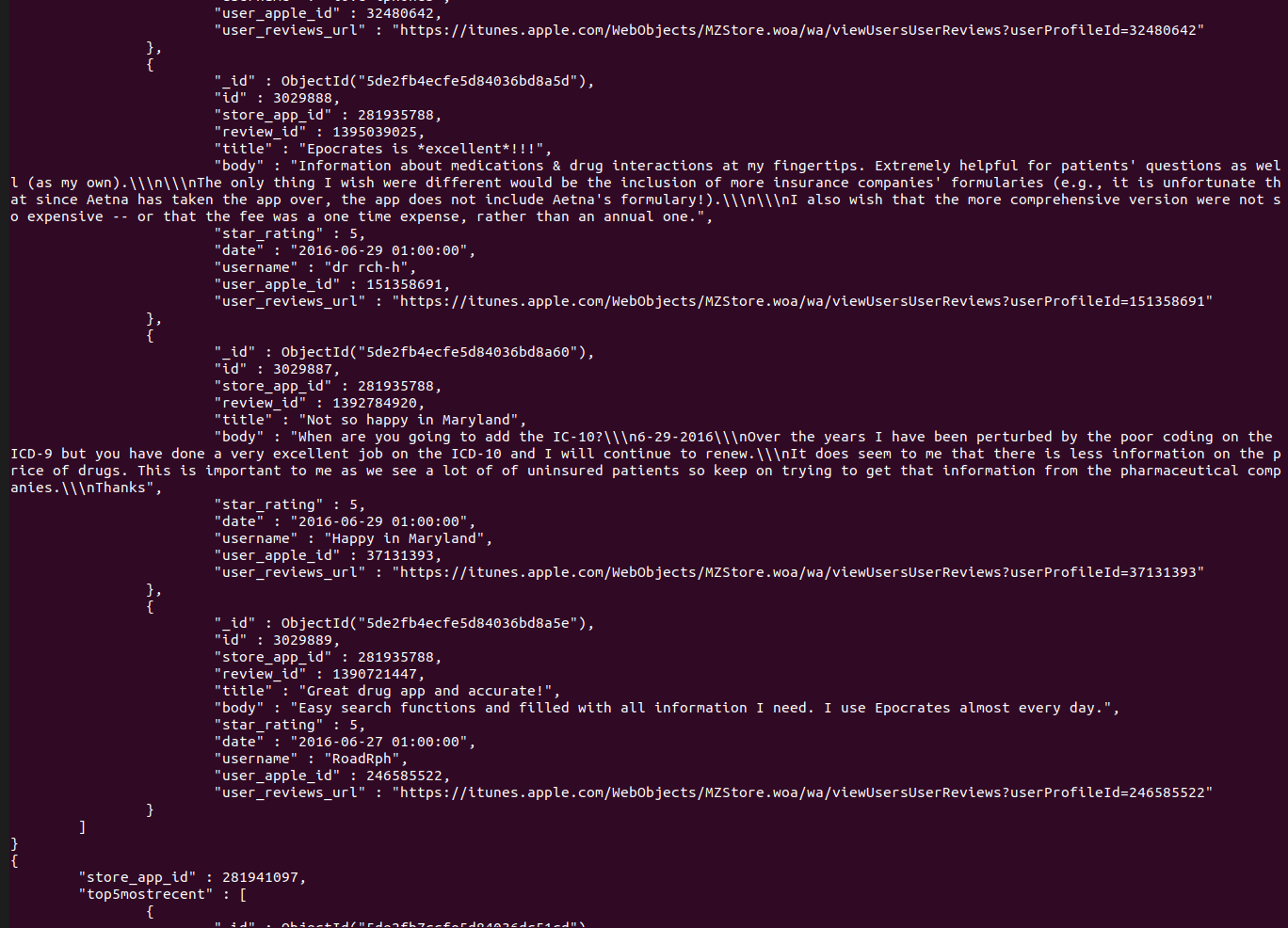
Justification: A histogram representation of the number/proportion of reviews is a great way to provide information at a glance. Just by looking at it we can see the numbers of reviews received per rating bucket (here in our case 1 through 5) and also their proportion to each other. With this being displayed at the top of an application’s page we can get the most important matrices related to the application easily.

#### Top 5 most recent reviews

db.reviews.aggregate([{$sort: {"date": -1}}, {$group: {\_id: "$store\_app\_id", reviews: {$push: "$$ROOT"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", top5mostrecent: {$slice: ["$reviews", 5]}}}], {allowDiskUse: true}).pretty()

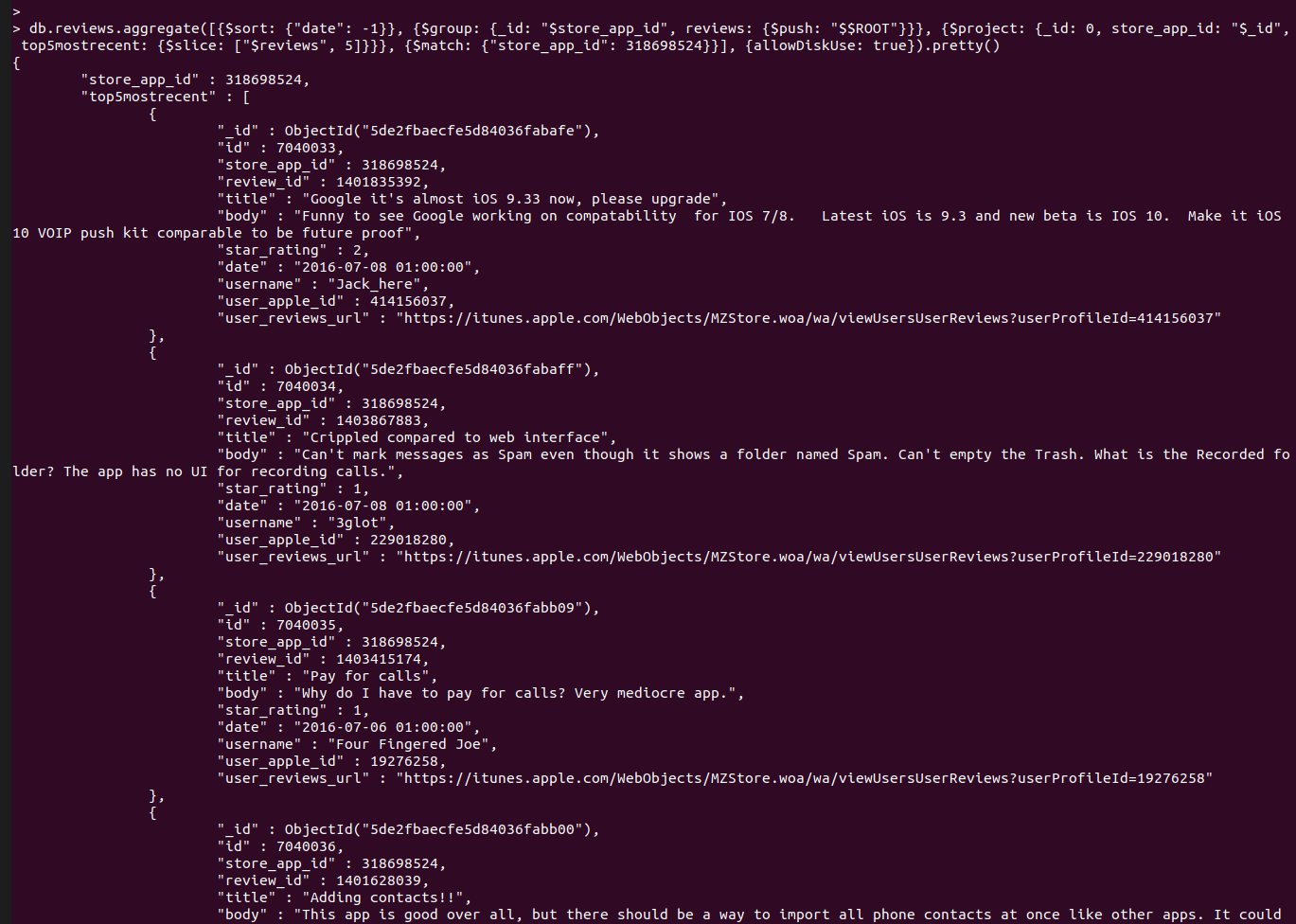


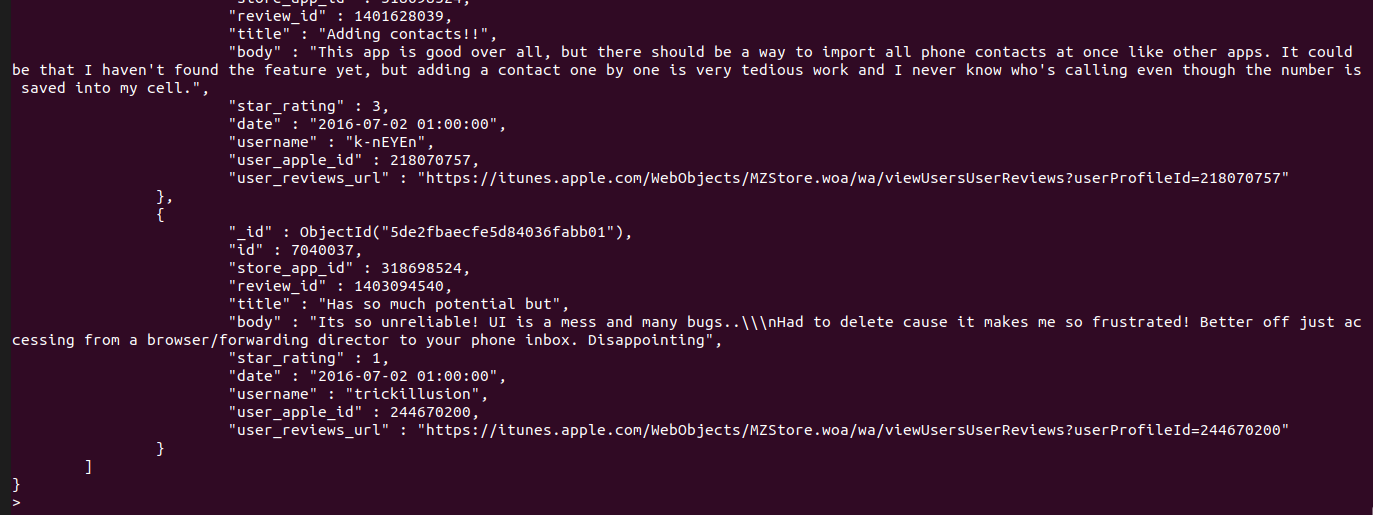




For Google Voice:

db.reviews.aggregate([{$sort: {"date": -1}}, {$group: {\_id: "$store\_app\_id", reviews: {$push: "$$ROOT"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", top5mostrecent: {$slice: ["$reviews", 5]}}}, {$match: {"store\_app\_id": 318698524}}], {allowDiskUse: true}).pretty()





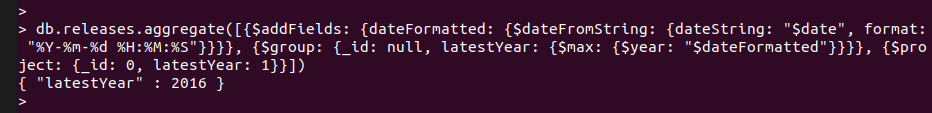
Summary: Above in the screenshots, we can see the 5 most recent reviews, dating between 2016-07-02 and 07-08

Justification: Seeing some of the most recent reviews gives an idea about the application's performance for the current release. With the dates of the reviews displayed we can also see if there are users who submitted ratings to the application recently, this might give a popularity sense to the app: popular application receive lots of reviews all the time, less popular ones might have infrequent reviews.

#### Number of releases this year

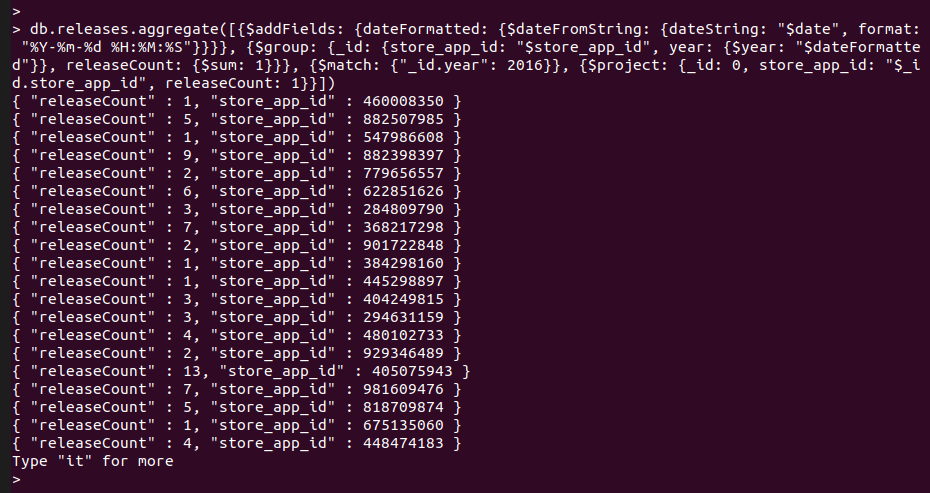
Note: To know what is “this year” in terms of the dataset:

db.releases.aggregate([{$addFields: {dateFormatted: {$dateFromString: {dateString: "$date", format: "%Y-%m-%d %H:%M:%S"}}}}, {$group: {\_id: null, latestYear: {$max: {$year: "$dateFormatted"}}}}, {$project: {\_id: 0, latestYear: 1}}])



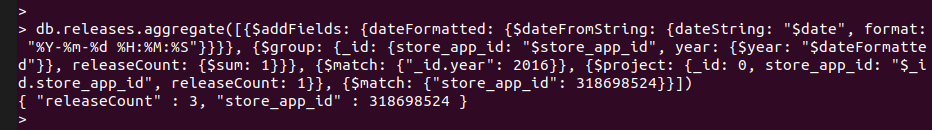
Note: Since the latest year in the releases collection is 2016, we are using 2016 as the current year

db.releases.aggregate([{$addFields: {dateFormatted: {$dateFromString: {dateString: "$date", format: "%Y-%m-%d %H:%M:%S"}}}}, {$group: {\_id: {store\_app\_id: "$store\_app\_id", year: {$year: "$dateFormatted"}}, releaseCount: {$sum: 1}}}, {$match: {"\_id.year": 2016}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", releaseCount: 1}}])



For Google Voice:

db.releases.aggregate([{$addFields: {dateFormatted: {$dateFromString: {dateString: "$date", format: "%Y-%m-%d %H:%M:%S"}}}}, {$group: {\_id: {store\_app\_id: "$store\_app\_id", year: {$year: "$dateFormatted"}}, releaseCount: {$sum: 1}}}, {$match: {"\_id.year": 2016}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", releaseCount: 1}}, {$match: {"store\_app\_id": 318698524}}])

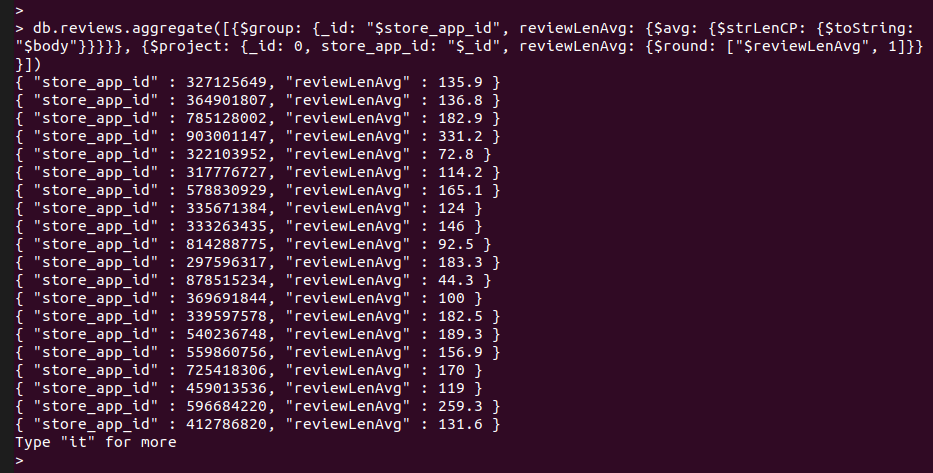


Summary: This year (2016) Google Voice had 3 updates published.

Justification: As a user of an application it would be good to know how often an application is updated, this way the user can be assured that the bugs are found and removed from the system. When updating applications we do not really check which applications are being updated, and we definitely do not remember which were apps updated a month ago. With this additional metric, a user just by opening the page of an app could see if it is an actively maintained application or not even before installing it.

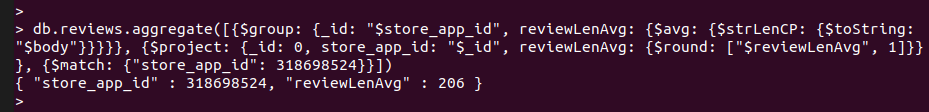
#### Average length of reviews

db.reviews.aggregate([{$group: {\_id: "$store\_app\_id", reviewLenAvg: {$avg: {$strLenCP: {$toString: "$body"}}}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", reviewLenAvg: {$round: ["$reviewLenAvg", 1]}}}])



For Google Voice:

db.reviews.aggregate([{$group: {\_id: "$store\_app\_id", reviewLenAvg: {$avg: {$strLenCP: {$toString: "$body"}}}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", reviewLenAvg: {$round: ["$reviewLenAvg", 1]}}}, {$match: {"store\_app\_id": 318698524}}])

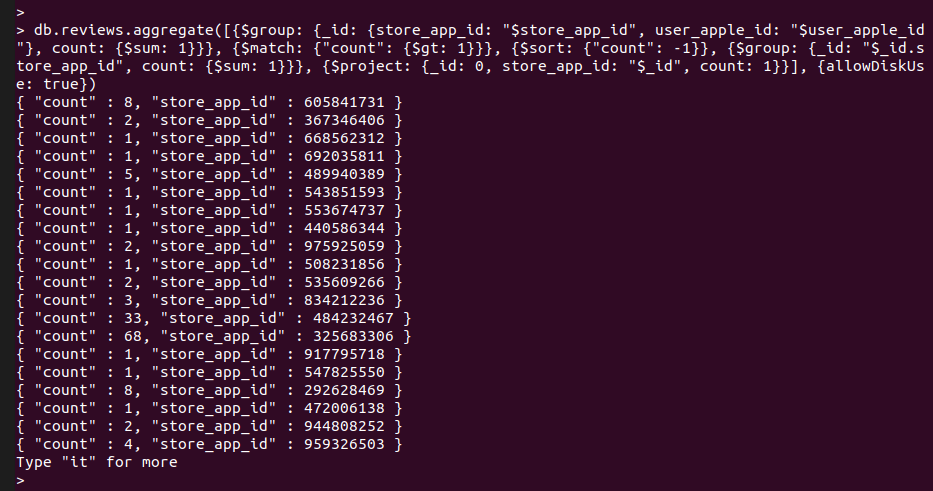


Summary: For Google Voice the average review length is 206 characters, meaning that most reviewers are expressing their thoughts not just through a simple star rating but they are providing helpful feedback as well based on which Google could improve its app published on the Apple App Store.

Justification: Seeing the average length of the reviews gives a sense of user involvement in the life of an app. Reviewers who write long reviews want the app to develop to a level where it is satisfactory. For example, if you see an app with an average review length of ~10 characters you can be assured that the users do not care that much about the app so they will just write a really short review to get rid of the notification of “Review the app if you like it”, while a review length of ~100 as where the reviews might include suggestions as well.

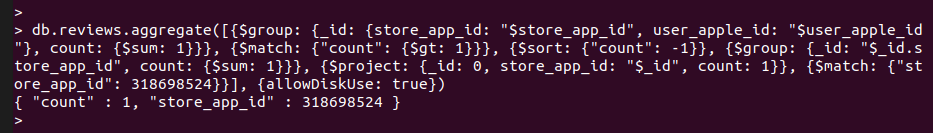
#### How many reviewers reviewed more than once?

db.reviews.aggregate([{$group: {\_id: {store\_app\_id: "$store\_app\_id", user\_apple\_id: "$user\_apple\_id"}, count: {$sum: 1}}}, {$match: {"count": {$gt: 1}}}, {$sort: {"count": -1}}, {$group: {\_id: "$\_id.store\_app\_id", count: {$sum: 1}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", count: 1}}], {allowDiskUse: true})



For Google Voice:

db.reviews.aggregate([{$group: {\_id: {store\_app\_id: "$store\_app\_id", user\_apple\_id: "$user\_apple\_id"}, count: {$sum: 1}}}, {$match: {"count": {$gt: 1}}}, {$sort: {"count": -1}}, {$group: {\_id: "$\_id.store\_app\_id", count: {$sum: 1}}}, {$project: {\_id: 0, store\_app\_id: "$\_id", count: 1}}, {$match: {"store\_app\_id": 318698524}}], {allowDiskUse: true})

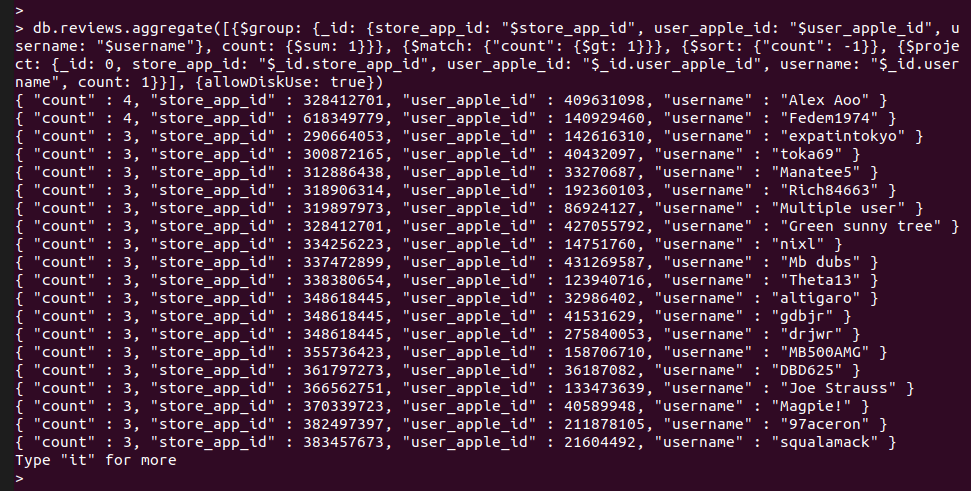


Summary: For Google Voice there is only one reviewer who reviewed the app more than once.

Justification: Since apps can request the users twice every year to submit a review, it is interesting to see how many reviewers review the application more than once. The 8th will be based on this metric as well.

#### Which reviewers reviewed more than once?

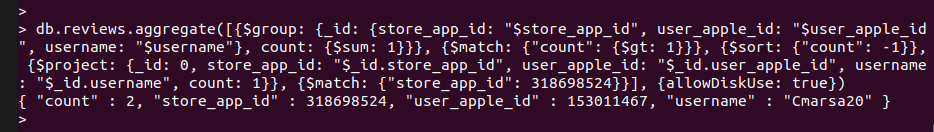
db.reviews.aggregate([{$group: {\_id: {store\_app\_id: "$store\_app\_id", user\_apple\_id: "$user\_apple\_id", username: "$username"}, count: {$sum: 1}}}, {$match: {"count": {$gt: 1}}}, {$sort: {"count": -1}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", user\_apple\_id: "$\_id.user\_apple\_id", username: "$\_id.username", count: 1}}], {allowDiskUse: true})



Summary: It seems like that the maximum number of times users have reviewed the same application is 4, and there is only 2 users who did that. Reviewing the same app 3 times seems more frequent.

For Google Voice:

db.reviews.aggregate([{$group: {\_id: {store\_app\_id: "$store\_app\_id", user\_apple\_id: "$user\_apple\_id", username: "$username"}, count: {$sum: 1}}}, {$match: {"count": {$gt: 1}}}, {$sort: {"count": -1}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", user\_apple\_id: "$\_id.user\_apple\_id", username: "$\_id.username", count: 1}}, {$match: {"store\_app\_id": 318698524}}], {allowDiskUse: true})



Summary: The only user who reviewed more than once is the user with ID 153011467 and username of Cmarsa20 and he/she reviewed twice.

Justification: With this information as developers or owners we could prioritize the users, meaning that those users who take the time to submit more reviews are more loyal, so we should pay extra attention to them. We can even display their name as “Most Loyal user” on our page.

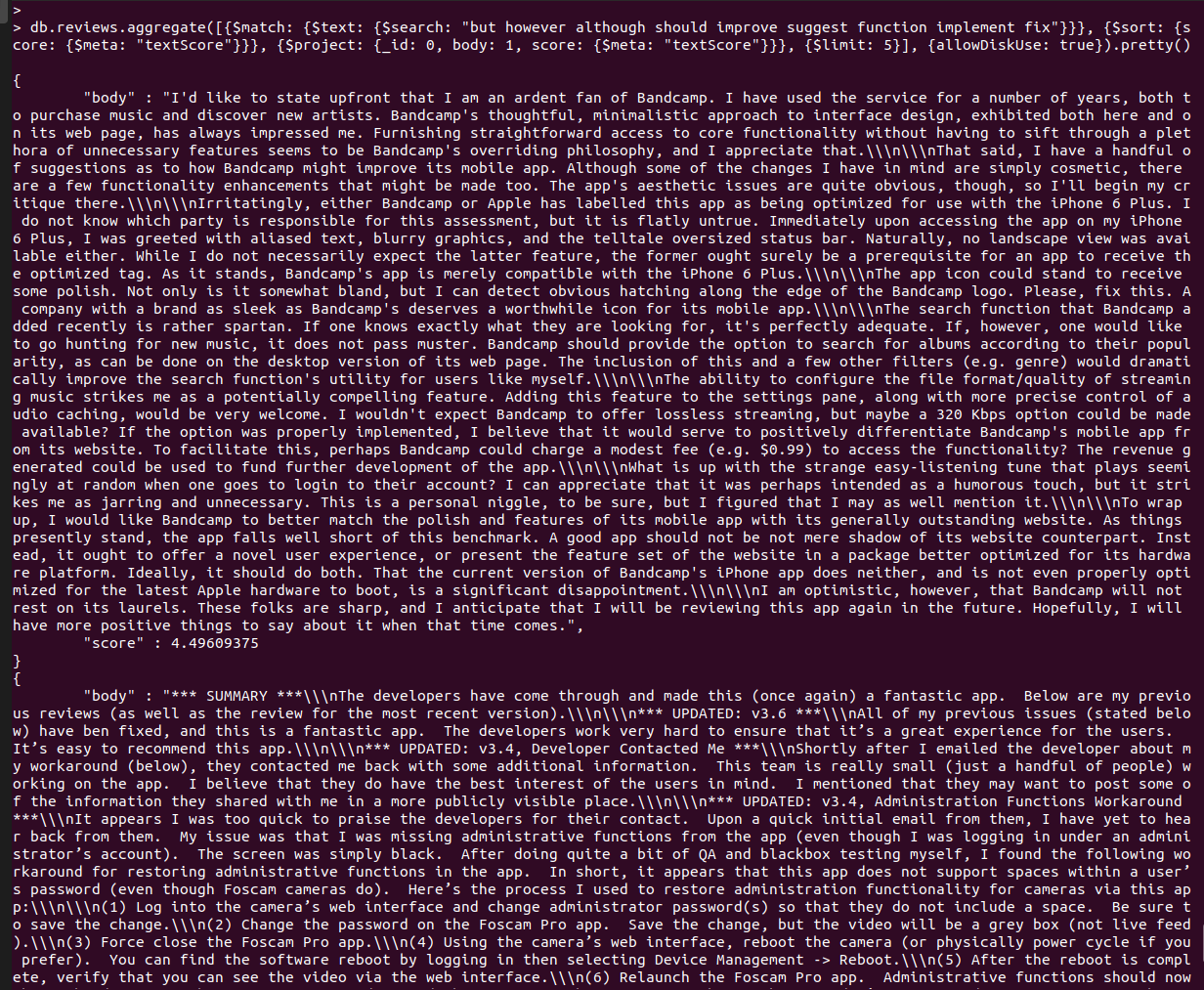
### (MongoDB) The issue of identifying the helpfulness of reviews is always tricky. Propose and derive a text-based definition. Run the queries and derive the results for both.

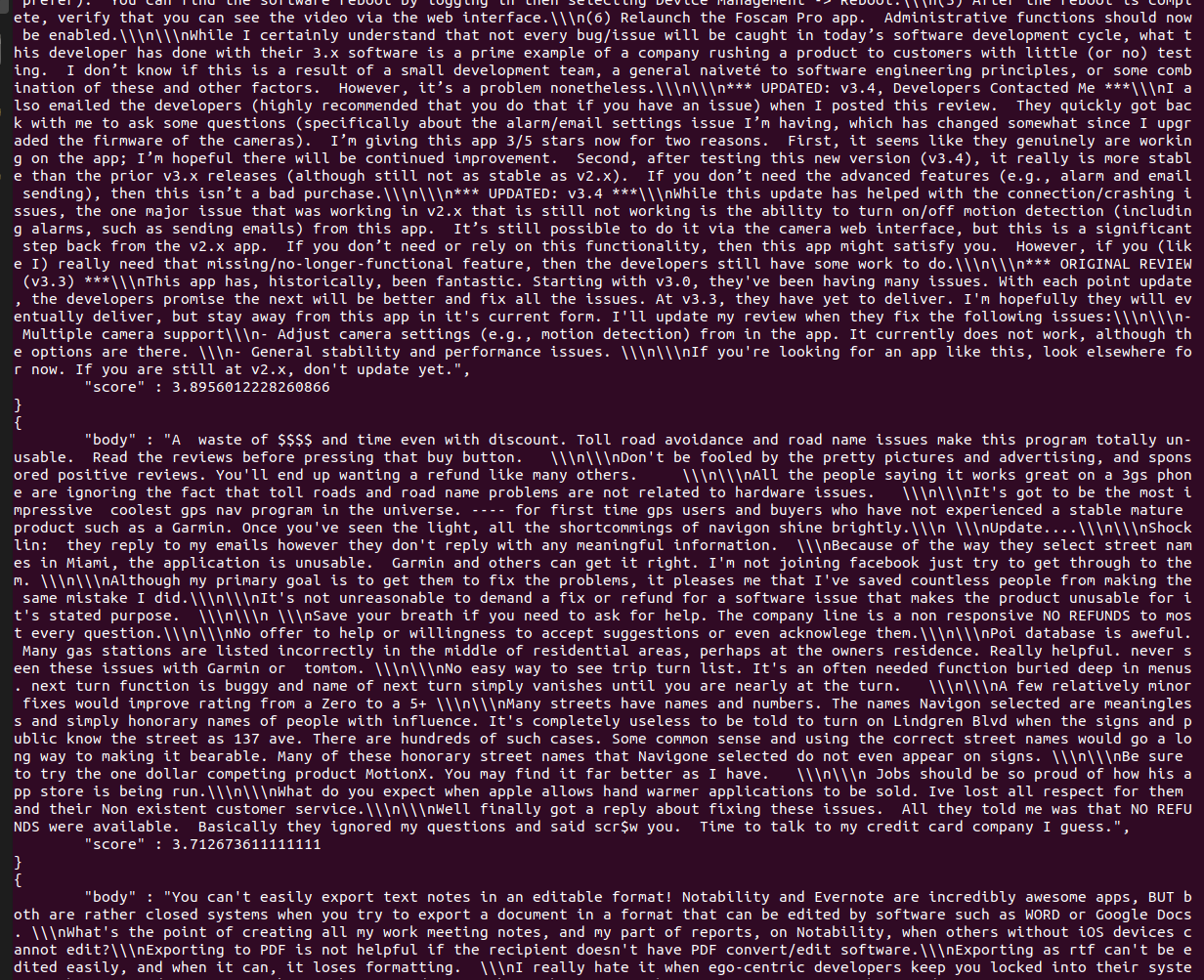
#### MongoDB index

db.reviews.createIndex({body: "text"})

Helpfulness: In this solution we are trying to find (or sort) reviews that are helpful for the developers and owners. The query is made towards words that can be used in connection with helpful ideas, suggestions that can be provided by the users for us.

db.reviews.aggregate([{$match: {$text: {$search: "but however although should improve suggest function implement fix"}}}, {$sort: {score: {$meta: "textScore"}}}, {$project: {\_id: 0, body: 1, score: {$meta: "textScore"}}}, {$limit: 5}], {allowDiskUse: true}).pretty()

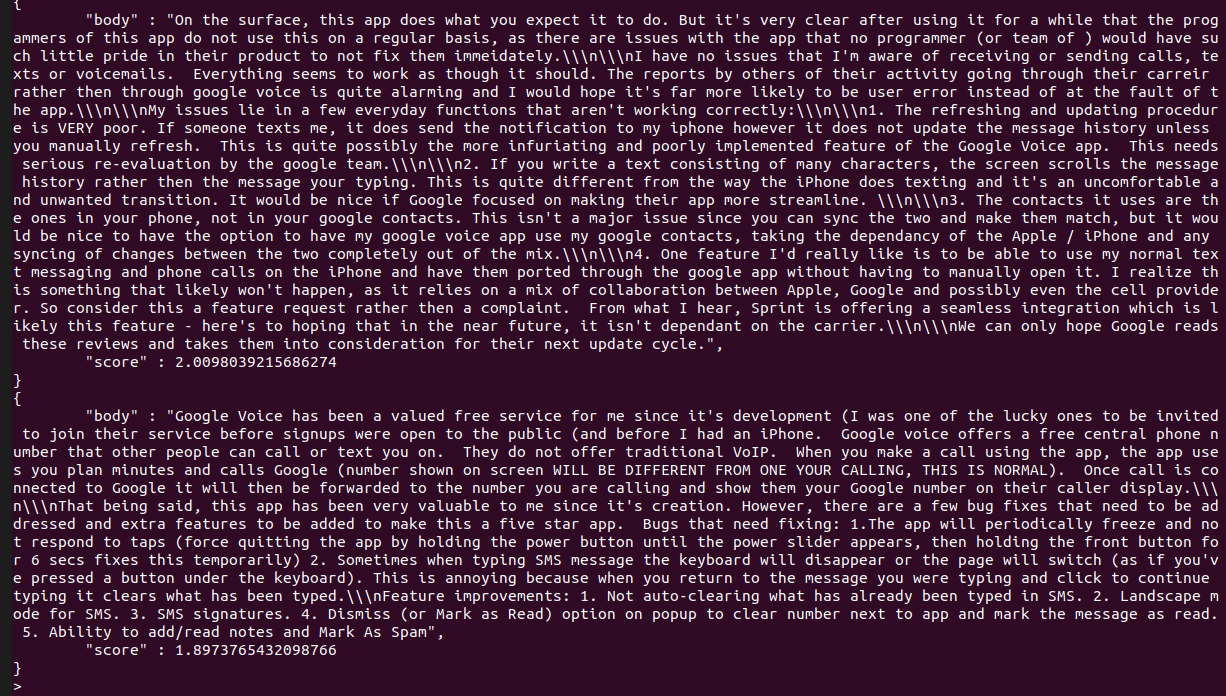




For Google Voice:

db.reviews.aggregate([{$match: {$text: {$search: "but however should improve suggest function implement fix"}}}, {$sort: {score: {$meta: "textScore"}}}, {$match: {"store\_app\_id": 318698524}}, {$project: {\_id: 0, body: 1, score: {$meta: "textScore"}}}, {$limit: 5}], {allowDiskUse: true}).pretty()



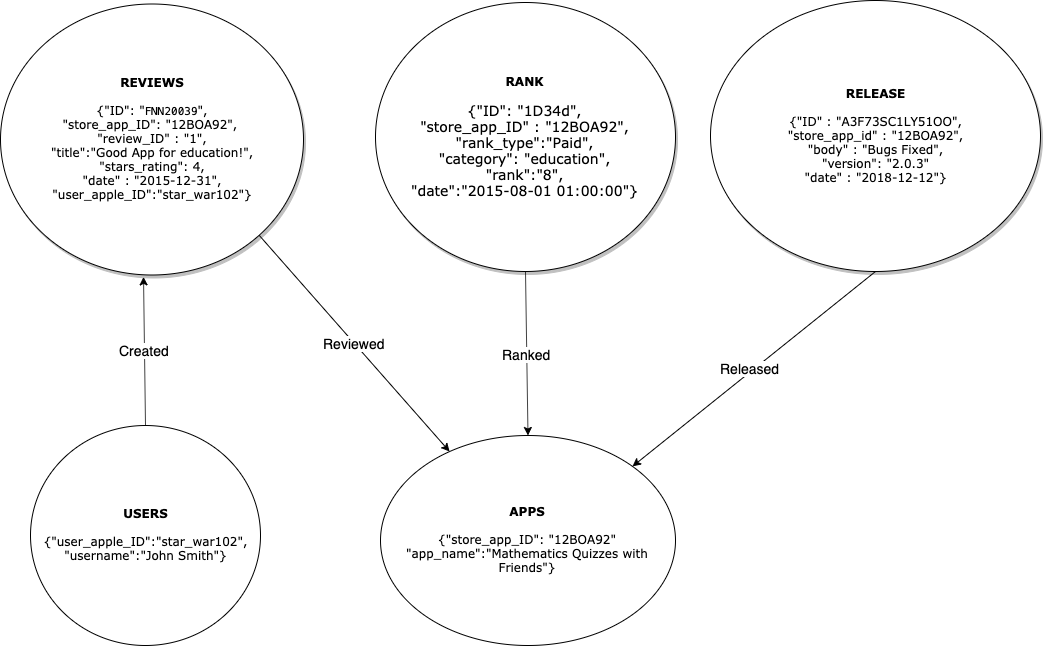


Summary: We are listing only the first 5 most helpful reviews due to space considerations. Above in the screenshots, you can read the 5 most helpful reviews about Google Voice that can help Google developers improve the user experience when using the application, and since the user is the most important asset, they will need to consider their reviews for making changes.

Justification: Contrasting words like ***but***, ***however***, ***although*** are words we use when we want to express how something should be. This also brings the word ***should*** as another word we use when ***suggest***ing an ***improve***ment or to ask the developers to ***fix*** an issue or ***implement*** a ***function***ality. These highlighted words are the ones we have decided that they carry high importance for the developers and owners of the application, users would probably use these words when expressing their helpful thoughts towards asking for an improvement in an app.

## Neo4j

### Simplified Design



### Import commands

mongo

use apple

db.createView("usersView", "reviews", [{$group: {\_id: {user\_apple\_id: "$user\_apple\_id", username: "$username"}}}, {$project: {\_id: 0, user\_apple\_id: "$\_id.user\_apple\_id", username: "$\_id.username"}}])

db.usersView.aggregate([{$out: "users"}], { allowDiskUse: true })

db.createView("apps", "rankings", [{$group: {\_id: {store\_app\_id: "$store\_app\_id", app\_name: "$app\_name"}}}, {$project: {\_id: 0, store\_app\_id: "$\_id.store\_app\_id", app\_name: "$\_id.app\_name"}}])

mongoexport --db apple --collection apps --type=csv --fields store\_app\_id,app\_name --out /var/opt/apps.csv

mongoexport --db apple --collection users --type=csv --fields user\_apple\_id,username --out /var/opt/users.csv

mongoexport --db apple --collection rankings --type=csv --fields id,store\_app\_id,rank\_type,category,rank,date --out /var/opt/rankings.csv

mongoexport --db apple --collection releases --type=csv --fields id,store\_app\_id,body,version,date --out /var/opt/releases.csv

mongoexport --db apple --collection reviews --type=csv --fields id,store\_app\_id,review\_id,title,body,star\_rating,date,user\_apple\_id,user\_reviews\_url --out /var/opt/reviews.csv

#smaller

mongoexport --db apple --collection reviews --type=csv --fields id,store\_app\_id,review\_id,title,star\_rating,date,user\_apple\_id --out /var/opt/reviewsSmaller.csv

USING PERIODIC COMMIT 500

LOAD CSV WITH HEADERS FROM "file:///apps.csv" AS row

CREATE (m:App {store\_app\_id: row.store\_app\_id, app\_name: row.app\_name})

USING PERIODIC COMMIT 500

LOAD CSV WITH HEADERS FROM "file:///users.csv" AS row

CREATE (m:User {user\_apple\_id: row.user\_apple\_id, username: row.username})

USING PERIODIC COMMIT 500

LOAD CSV WITH HEADERS FROM "file:///rankings.csv" AS row

CREATE (m:Ranking {id: row.id, store\_app\_id: row.store\_app\_id, rank\_type: row.rank\_type, category: row.category, rank: row.rank, date: row.date})

USING PERIODIC COMMIT 500

LOAD CSV WITH HEADERS FROM "file:///releases.csv" AS row

CREATE (m:Release {id: row.id, store\_app\_id: row.store\_app\_id, body: row.body, version: row.version, date: row.date})

USING PERIODIC COMMIT 500

LOAD CSV WITH HEADERS FROM "file:///reviewsSmaller.csv" AS row

CREATE (m:Review {id: row.id, store\_app\_id: row.store\_app\_id, review\_id: row.review\_id, title: row.title, star\_rating: toInteger(row.star\_rating), date: row.date, user\_apple\_id: row.user\_apple\_id})

MATCH (a:Ranking), (b:App)

WHERE a.store\_app\_id = b.store\_app\_id

CREATE (a)-[r:RANKED]->(b)

MATCH (a:Release), (b:App)

WHERE a.store\_app\_id = b.store\_app\_id

CREATE (a)-[r:RELEASED]->(b)

#install apoc plugin

CALL apoc.periodic.iterate(

"MATCH (a:User), (b:Review)

WHERE a.user\_apple\_id = b.user\_apple\_id RETURN a, b",

"CREATE (a)-[r:CREATED]->(b)",

{batchSize:100, parallel:true})

CALL apoc.periodic.iterate(

"MATCH (a:Review), (b:App)

WHERE a.store\_app\_id = b.store\_app\_id RETURN a, b",

"CREATE (a)-[r:REVIEWED]->(b)",

{batchSize:100, parallel:true})

### (Neo4j) Reviewers sometimes review as a mob (reviewing the common set of apps). Do you find this behavior in this dataset? Run the queries and derive the results.

MATCH (reviewer:User)-[:CREATED]->(review:Review)-[:REVIEWED]->(apl:App)

WITH reviewer AS reviewers, collect(apl) AS apps

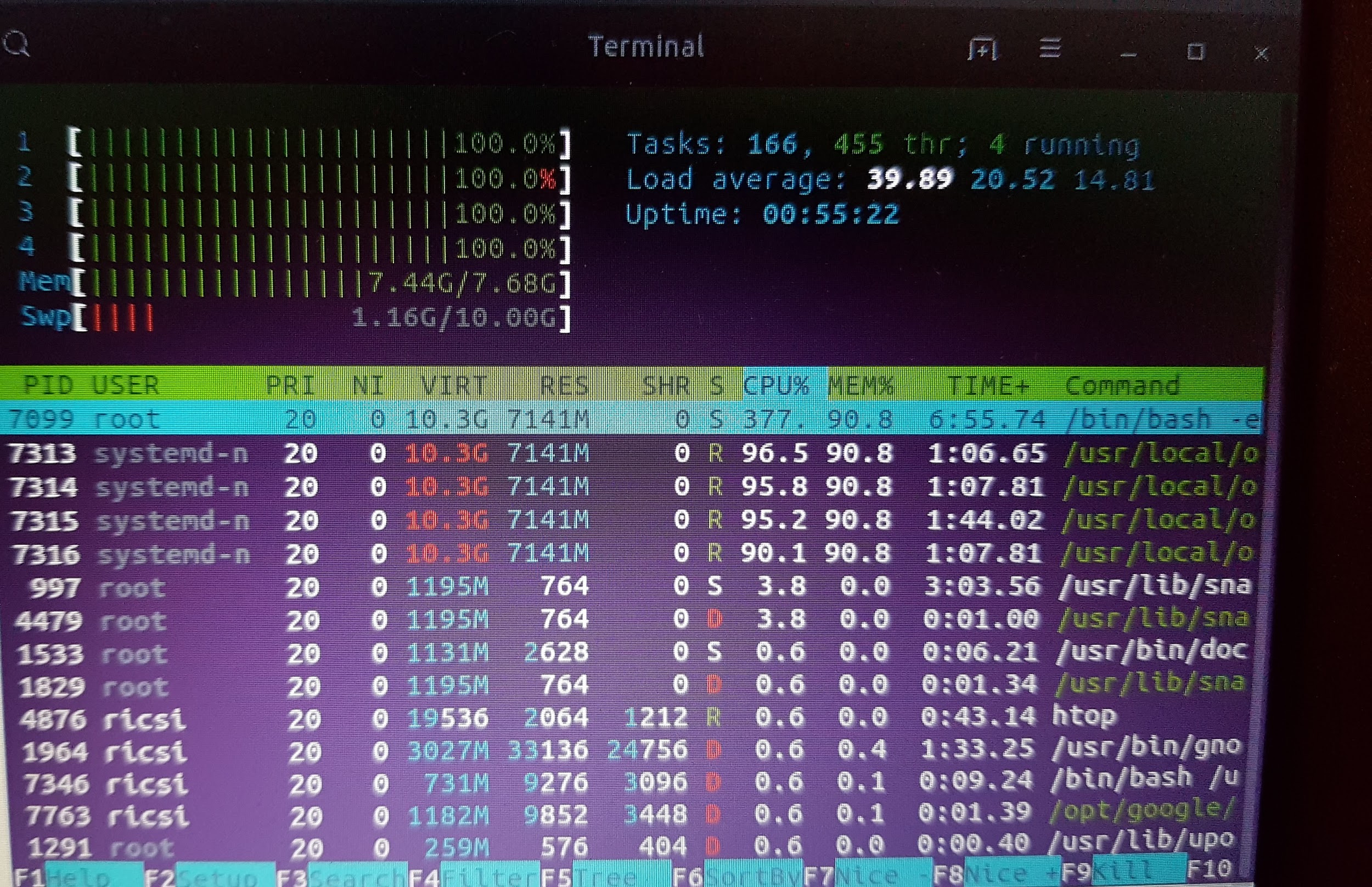
WITH apps, collect(reviewers) AS gang, count(reviewers) AS countReviewers

WHERE countReviewers > 2

RETURN apps, gang

ORDER BY countReviewers DESC

LIMIT 20



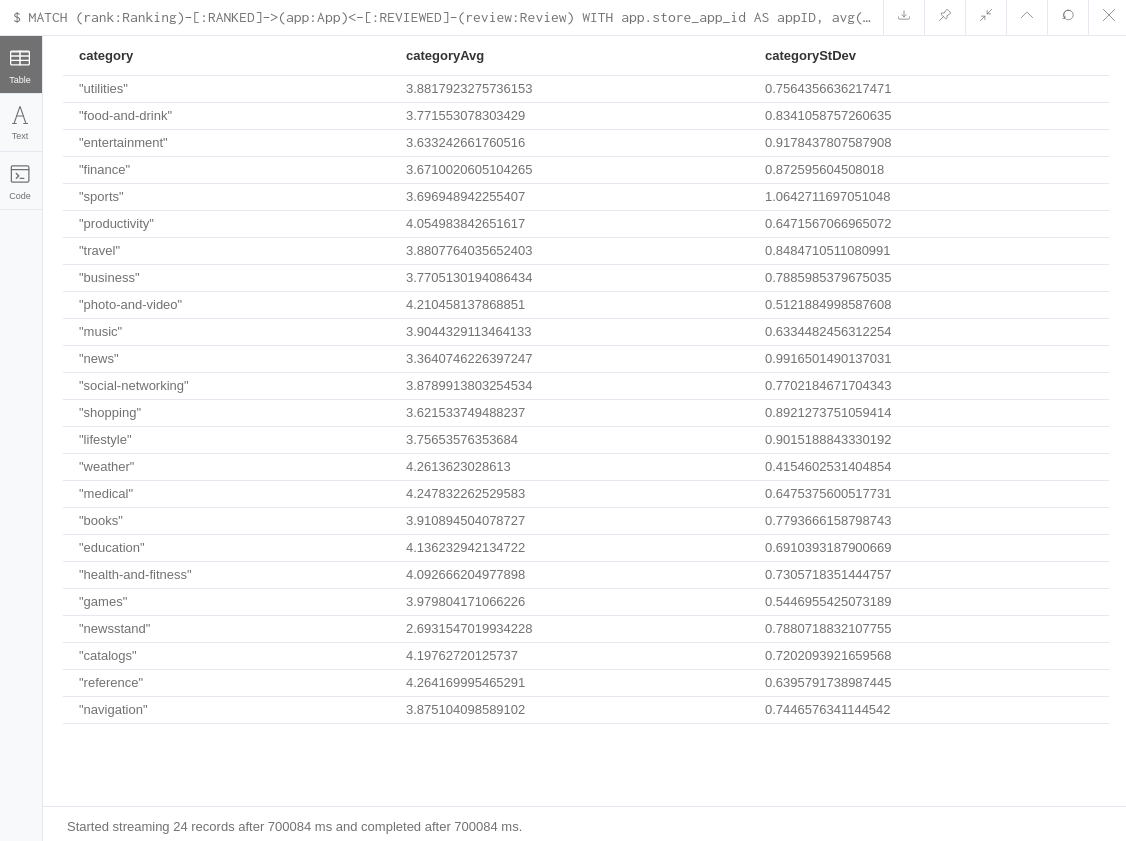
Summary: Unfortunately due to system memory size and processor load this query cannot be completed on a system with 8GBs of RAM. The picture shows how the query performed on an Intel i7 quad 3.4GHz and 8GB RAM with 10GB SSD swap.

### (Neo4j) Does similarity in app functionality lead to uniformity in app ratings or divergence? Run the queries and derive the results. Explain your findings.

MATCH (rank:Ranking)-[:RANKED]->(app:App)<-[:REVIEWED]-(review:Review)

WITH app.store\_app\_id AS appID, avg(review.star\_rating) AS appAvg, rank.category AS category

RETURN category, avg(appAvg) AS categoryAvg, stDevP(appAvg) AS categoryStDev



Summary:

For the explanation, we decided that where the standard deviation of the ratings is larger than 0.9, we can say that there is a high deviation so there is no uniformity at all. Where the standard deviation is between 0.6 and 0.9 there is a light uniformity, and below 0.6 we can say that there is a uniformity in the app ratings.

In the categories of entertainment, sports, news, and lifestyle we definitely cannot see uniformity based on the above rule we have set up.

In the categories of photo-and-video, weather, and games we have found a uniformity (based on the rule above).

### (SAS) The issue of identifying fake reviews is always tricky. Propose and derive a text-based definition and run the needed analysis. Explain your findings.

#### To export the data from MongoDB for text mining

mongoexport --db apple --collection reviews --type=csv --fields review\_id,title,body --out /var/opt/reviews\_text.csv

#### How to identify fake reviews?

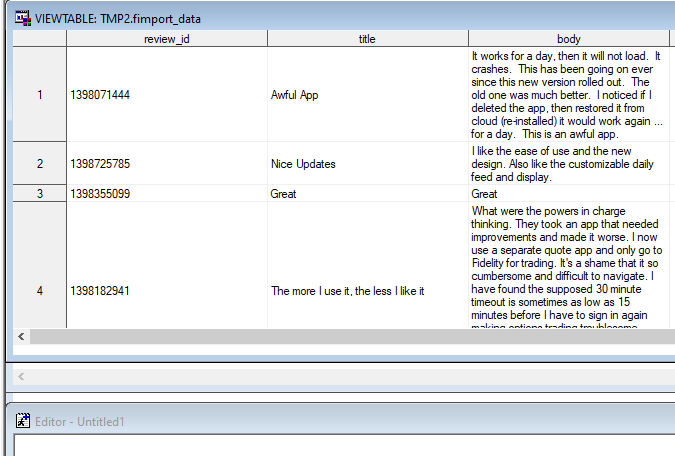
If there are reviews that have the same content they are likely to be fake, since there would be no 2 reviews worded exactly the same way.

#### Justification

Sample Size: 365 Rows

We believe that a review may be copied from one to another ,so their text content is about 90% or above the same. That is why in Text Cluster we use parameters: “Number of Clusters” : 180 ( ~ sample / 2) and “Exact or Maximum Number”: exact. We believe if two or three documents are almost the same, they will 100% found in the same cluster.

#### Input file

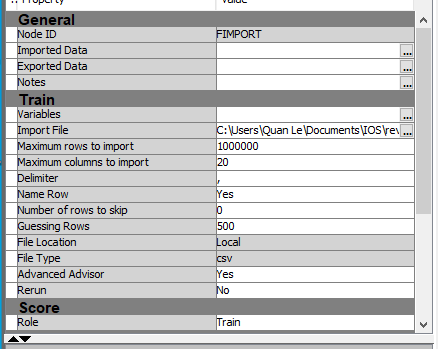


#### Workflow

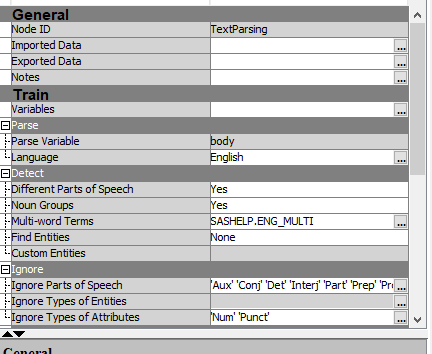


#### Parameters Chosen

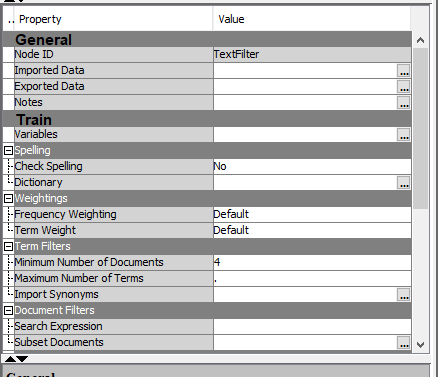
* File Import: Default



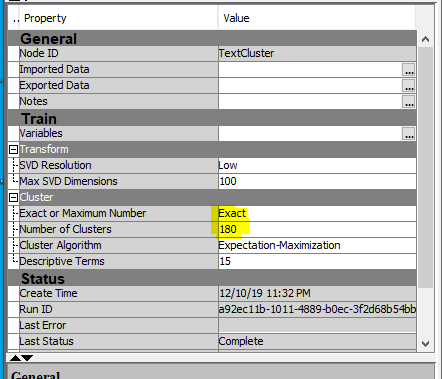
* Text Parsing: Default



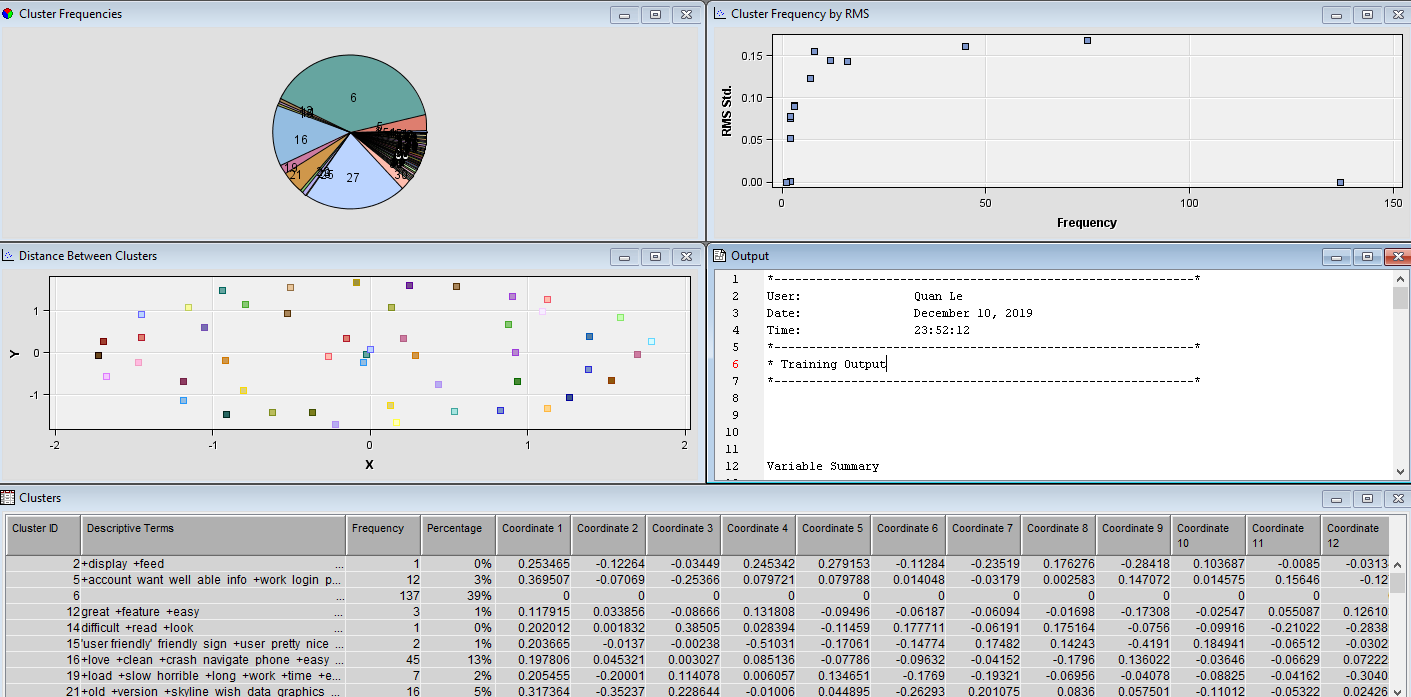
* Text: Filter: Default



* Text Cluster: Number of Clusters: 180 , Exact or Maximum Number: Exact



#### Output



#### Summary

We just need to check clusters that have more than one document because content identical problems involve at least two documents or above. Ignoring cluster 6 which is about “space”, cluster 21, 27, 16, 5, 12, and 19 have group of the same text cluster. After checking all of them, we don’t see any two content in those clusters that would be similar, so in our sample has no fake review.

#### Link to text clustering result

Sas table: <https://1drv.ms/u/s!Arj5DPebNUHkimstJWm4vj7Oy4uT?e=kz0r5q>

CSV file: <https://1drv.ms/u/s!Arj5DPebNUHkimwXteKiR5_xgh1P?e=yWFbkt>

# D - Design

# 