IMAT3104:

Database Management and Programming Assignment

LAMPROS KARSERAS  
P2424629

Table of Contents

[Q1: 3](#_Toc67649241)

[Q2: 4](#_Toc67649242)

[Q3: 5](#_Toc67649243)

[Q4: 6](#_Toc67649244)

[Q5: 7](#_Toc67649245)

[Q6: 8](#_Toc67649246)

[Q7: 9](#_Toc67649247)

[Q8: 10](#_Toc67649248)

[Q9: 11](#_Toc67649249)

[E1: 12](#_Toc67649250)

[D1: 14](#_Toc67649251)

# Q1:

#### Find the description and price of the product entitled “Kid's Astronaut Hat”. No other product details are required.

db.getCollection("P2424629\_products").find(

{ "lk\_title": "Kid's Astronaut Hat" },

{ "lk\_description": 1, "lk\_price": 1, "\_id": 0 }

)

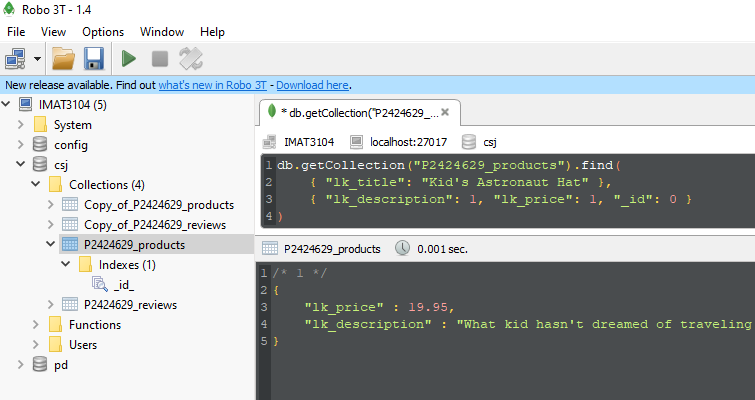


Figure 1

# Q2:

#### Find all details of products that are priced at either $7.99 or $14.99.

db.getCollection("P2424629\_products").find(

{ $or: [ { "lk\_price": 7.99 }, { "lk\_price": 14.99 } ] }

)

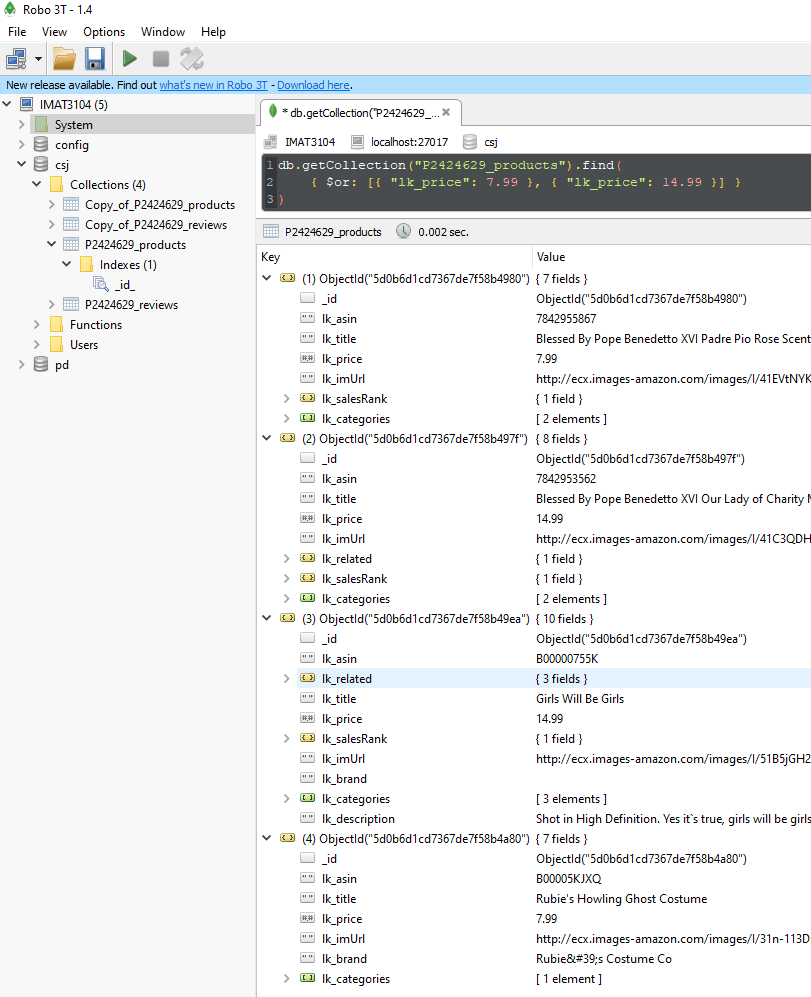


Figure 2

# Q3:

#### Count the number of products that have a price but do not have a description. Clearly show the number you receive as your answer.

db.getCollection("P2424629\_products").find(

{ "lk\_price": { $ne: null }, "lk\_description": null }

).count()

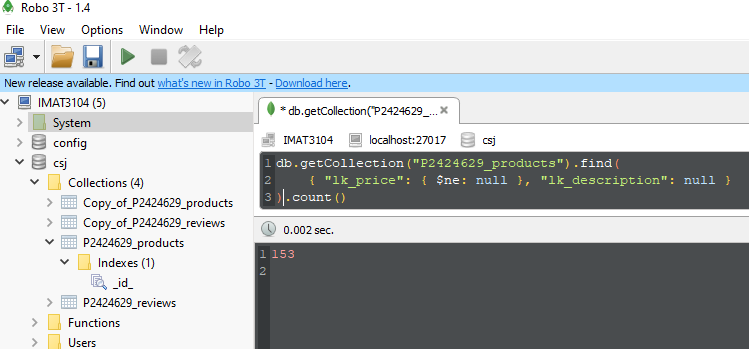


Figure 3

# Q4:

#### List the product IDs of products that were reviewed before the 15th June 2000. All calculations should be performed by MongoDB. Only list the product IDs and do not show any duplicates.

var filterTime = ISODate("2000-06-15").getTime() / 1000

db.P2424629\_reviews.aggregate([

{ $match: { lk\_unixReviewTime: { $lt: filterTime } } },

{ $group: { \_id: "$lk\_asin" } }

])

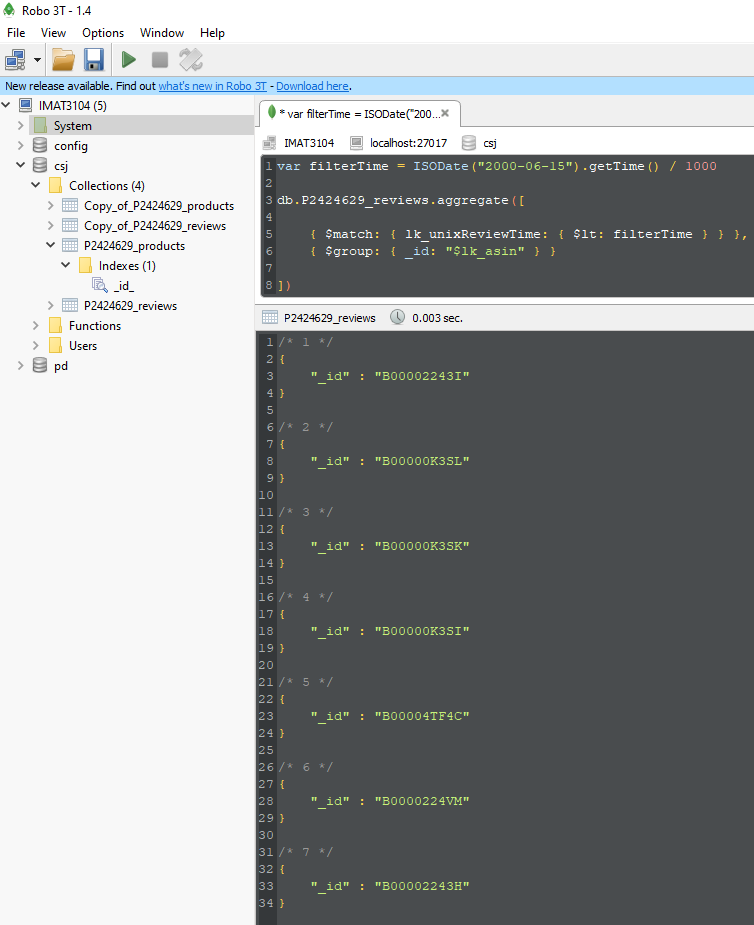


Figure 4

# Q5:

#### List all the reviewers who have written at least 3 reviews. Show the reviewer ID, reviewer name and the number of reviews for each one. List the names of reviewers in alphabetical order. [Hint: it is possible to solve using aggregate pipeline method].

db.getCollection("P2424629\_reviews").aggregate([

{

"$group": {

"\_id": "$lk\_reviewerID",

"NumberOfReviews": { "$sum": 1 },

"ReviewerName": { "$first": "$lk\_reviewerName" }

}

},

{ "$match": { "NumberOfReviews": { "$gte": 3 } } },

{ "$sort": { "ReviewerName": 1 } },

{

"$project": {

"\_id": 0,

"ReviewerID": "$\_id",

"ReviewerName": "$ReviewerName",

"NumberOfReviews": "$NumberOfReviews"

}

}])

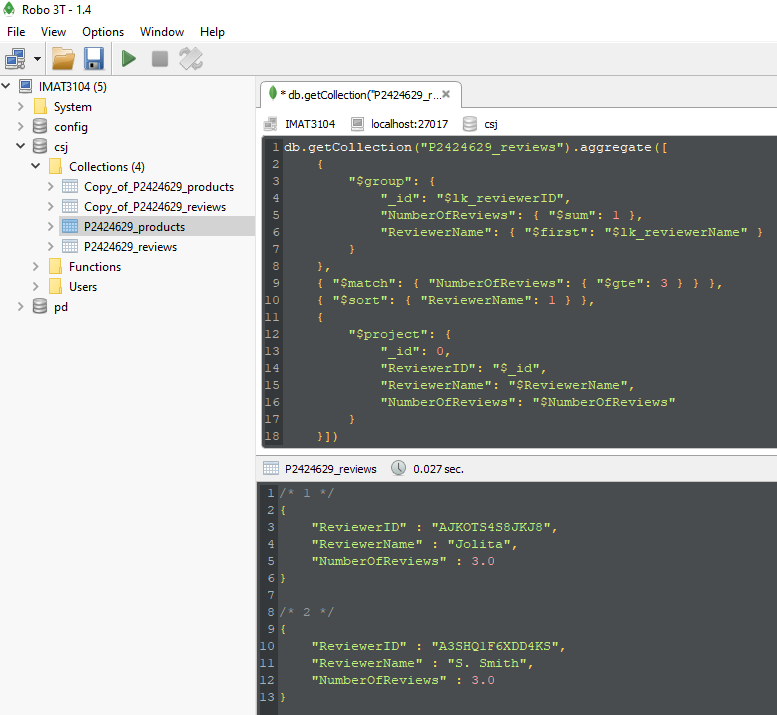


Figure 5

# Q6:

#### Count the number of related products that were also viewed with the product “Mens Timex”. Only give the number of related products as your answer. [Hint: it is possible to solve using aggregate pipeline method].

db.getCollection("P2424629\_products").aggregate([

{

$match: { lk\_title: "Mens Timex" }

},

{

$project: {

\_id: 0,

NumberOfRelatedViewed: { $size: "$lk\_related.also\_viewed" }

}

}

])



Figure 6

# Q7:

#### List the product ID and title of all products in the “T-Shirts” category. No other product details are required. [Hint: it is possible to solve using aggregate pipeline method].

db.getCollection("P2424629\_products").aggregate([

{ "$unwind": { "path": "$lk\_categories" } },

{ "$match": { "lk\_categories": "T-Shirts" } },

{

"$project": {

"\_id": 0,

"ProductID": "$lk\_asin",

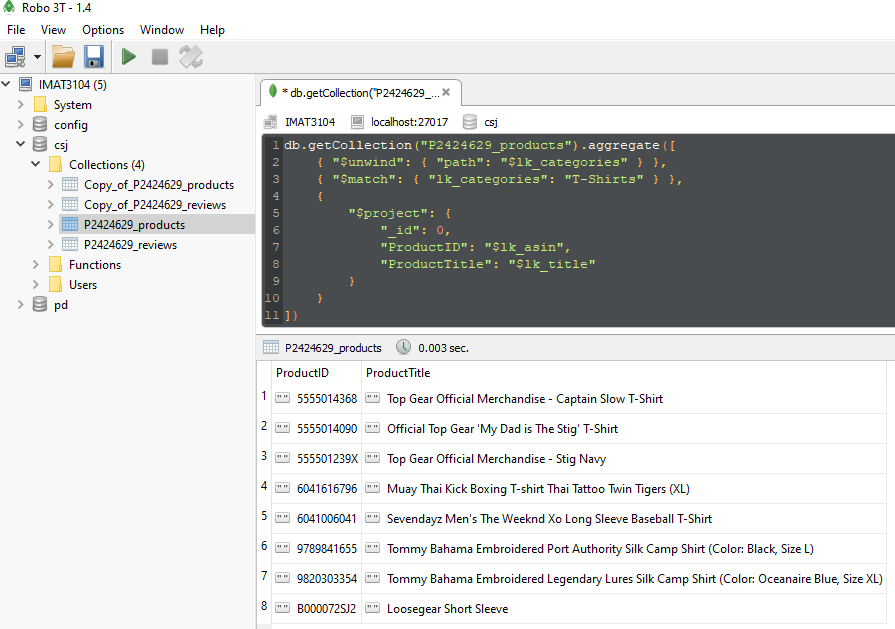
"ProductTitle": "$lk\_title"

}

}

])

Figure 7



# Q8:

#### Find the highest rating by the reviewer identified as “A2B3TVWTZ7609A” by using **map reduce** programming.

var map = function () {

emit(this.lk\_reviewerID, this.lk\_overall)

}

var reduce = function (key, values) {

var max = values[0]

values.forEach(function (value) {

if (value > max) {

max = value

}

})

return max

}

db.getCollection("P2424629\_reviews").mapReduce(map, reduce, {

query: { lk\_reviewerID: "A2B3TVWTZ7609A" },

out: { inline: 1 }

})

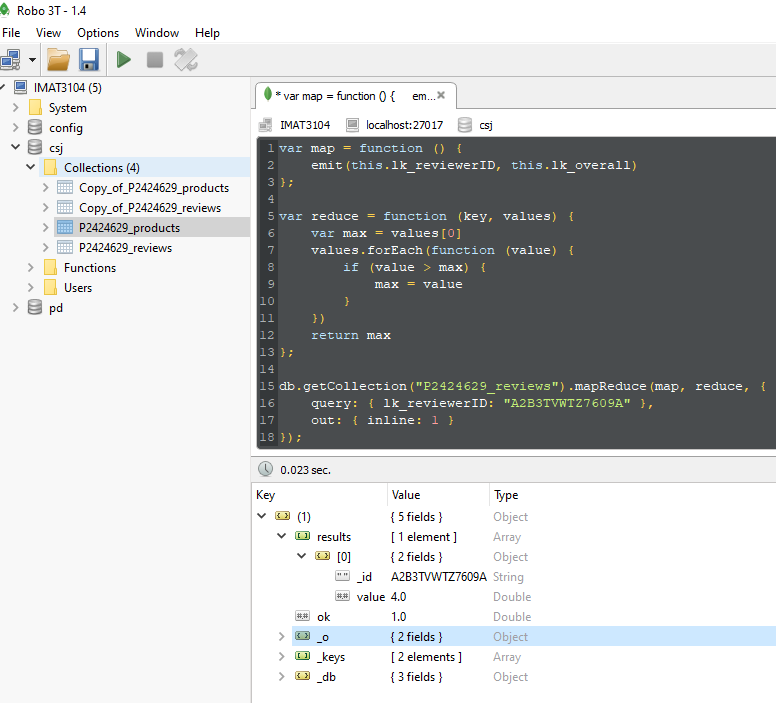


Figure 8

# Q9:

#### Using both collections, find the title and description of products reviewed by the reviewer identified by “A2C65Q9IDNCSR”. No other product details are required.

db.getCollection("P2424629\_reviews").aggregate([

{ $match: { lk\_reviewerID: "A2C65Q9IDNCSR" } },

{

$lookup: {

from: "P2424629\_products",

localField: "lk\_asin",

foreignField: "lk\_asin",

as: "product\_data"

}

},

{ $unwind: "$product\_data" },

{

$project: {

\_id: 0,

Title: "$product\_data.lk\_title",

Description: "$product\_data.lk\_description",

}

}

])

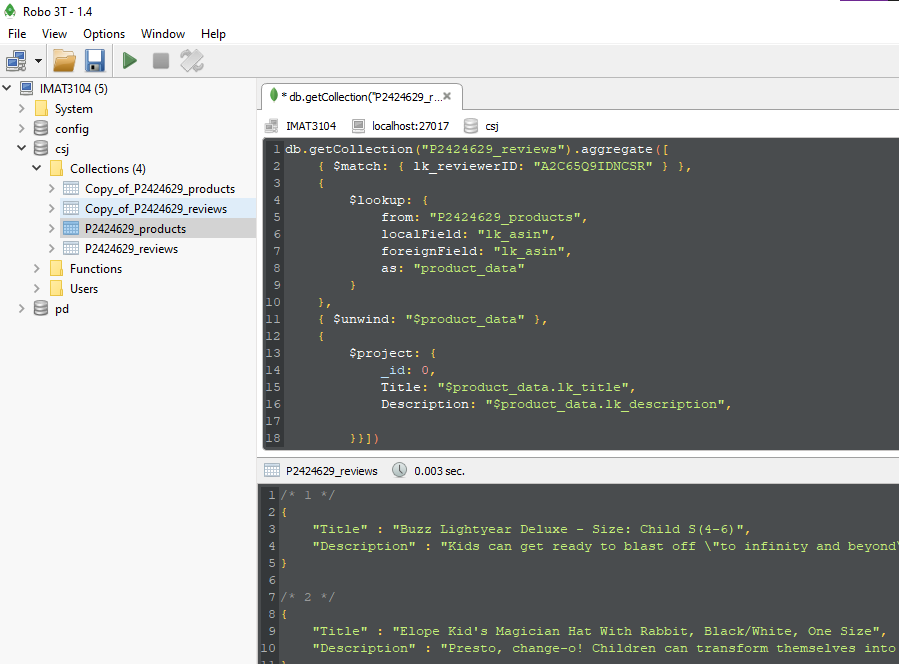


Figure 9

# E1:

#### Implement one index that would improve the querying of the database based on one or more of the queries (Q1-Q9). Identify the chosen query or queries and explain and justify your choice of index. Present, explain and compare execution plans to support your choice and summarise your findings

db.P2424629\_products.ensureIndex({lk\_title: 1})

My rational for choosing “lk\_title” field as the index for products is, the benefits in query speeds for queries Q1 and Q6 and for future proof database design.  
Even though the products collection only has 552 documents, this is just a sample of the extremely large Amazon DB, which consists of millions of products. The product’s titles will be used in significant number of queries in the future.  
Another reason for choosing the “lk\_title” field is the very high **cardinality**. Since every product is unique, the indexing of “lk\_title” field will yield significant gains in speed.  
**Selectivity** is another point in making the “lk\_title” field an index. It is, by default, a very selective field, which will return very few results if used.

For example, analysing the queries Q1 and Q6 with explain**(**"allPlansExecution"**)** shows the benefits of indexing the “lk\_title” field. As shown in Figure10, without indexing there is an InputStage of full collection Scan, which means the query “walked” through all the 552 documents to return 1, as shown in Figure11.

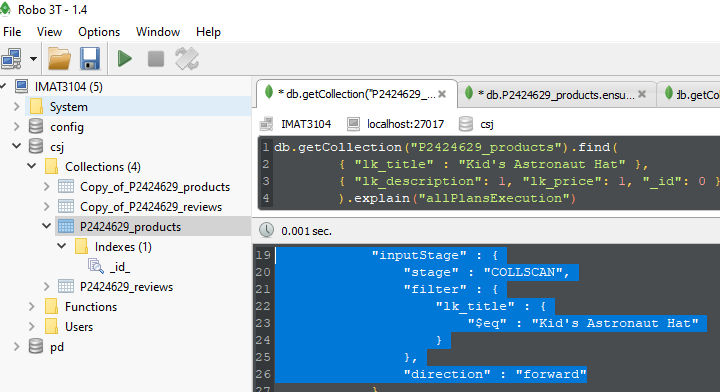


Figure 10

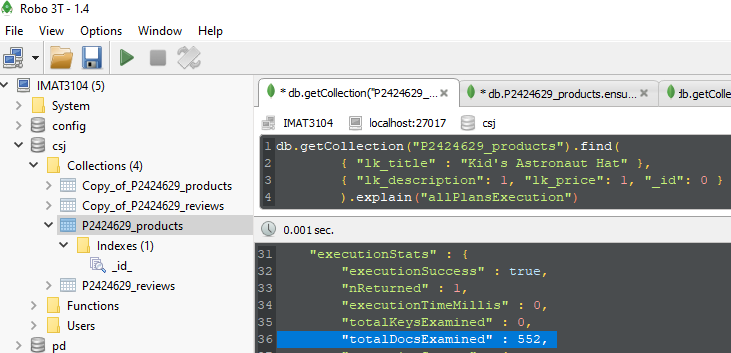


Figure 11

With indexing the “lk\_title” field, the query just immediately returns a single document, without examining all the collection, as show in Figure12. The input Stage is changed to FETCH and IXSCAN, which means it looks just in indexed entries, and examined just a single document.

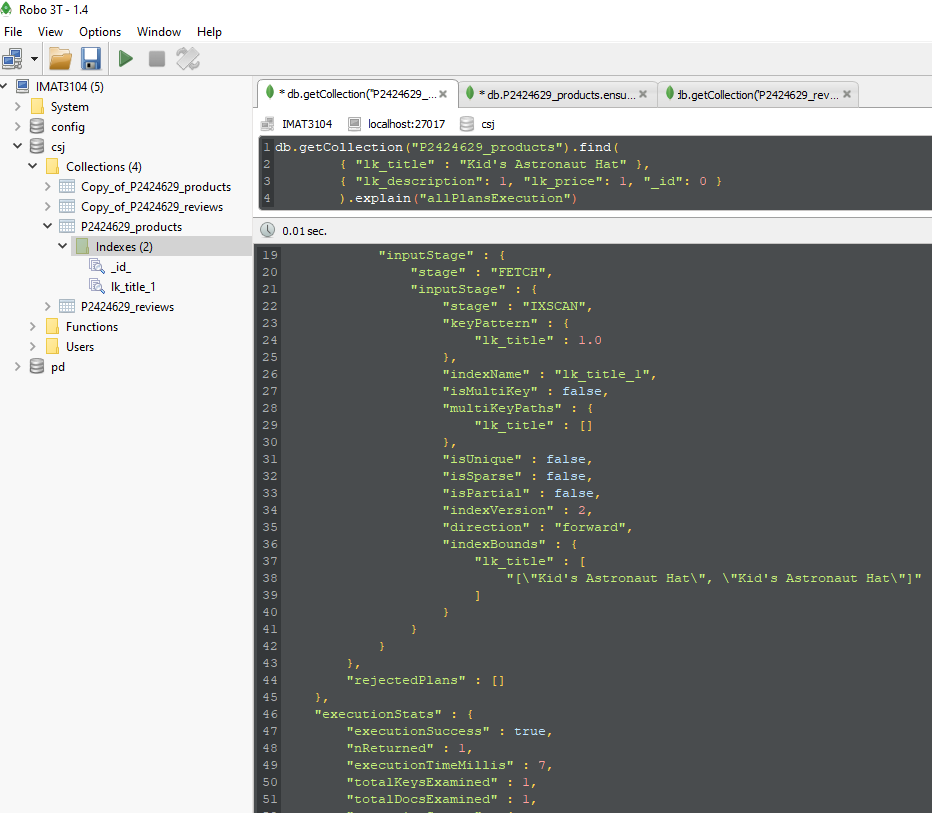


Figure 12

# D1:

#### Write code in MongoDB to automatically embed the details of reviews from the reviews collection with their corresponding product in the product collection.

#### Therefore, the products collection will contain both the product data and review data in a single collection of products.

var reviews\_cursor = db.Copy\_of\_P2424629\_reviews.find({}, { \_id: 0 })

while (reviews\_cursor.hasNext()) {

var nextReviewDoc = reviews\_cursor.next()

var review\_lk\_asin = nextReviewDoc.lk\_asin

var review\_lk\_reviewerID = nextReviewDoc.lk\_reviewerID

var review\_lk\_reviewerName = nextReviewDoc.lk\_reviewerName

var review\_lk\_helpful = nextReviewDoc.lk\_helpful

var review\_lk\_reviewText = nextReviewDoc.lk\_reviewText

var review\_lk\_overall = nextReviewDoc.lk\_overall

var review\_lk\_summary = nextReviewDoc.lk\_summary

var review\_lk\_unixReviewTime = nextReviewDoc.lk\_unixReviewTime

var review\_lk\_reviewTime = nextReviewDoc.lk\_reviewTime

db.Copy\_of\_P2424629\_products.update(

{ lk\_asin: review\_lk\_asin },

{

$push:

{

lk\_reviewerID: review\_lk\_reviewerID,

lk\_reviewerName: review\_lk\_reviewerName,

lk\_helpful: review\_lk\_helpful,

lk\_reviewText: review\_lk\_reviewText,

lk\_overall: review\_lk\_overall,

lk\_summary: review\_lk\_summary,

lk\_unixReviewTime: review\_lk\_unixReviewTime,

lk\_reviewTime: review\_lk\_reviewTime

}

})

}

// Query Testing with all the Review fields + lk\_title

db.Copy\_of\_P2424629\_products.find(

{ lk\_asin: "B00005JHK9" },

{

\_id: 0,

lk\_title: 1,

lk\_reviewerID: 1,

lk\_reviewerName: 1,

lk\_helpful: 1,

lk\_reviewText: 1,

lk\_overall: 1,

lk\_summary: 1,

lk\_unixReviewTime: 1,

lk\_reviewTime: 1

})

