Week 4: Document Databases

DIANE WOODBRIDGE, PH.D

MongoDB Interview Questions

MongoDB's type

MongoDB's characteristics

Alternative databases

Supported programming languages

Index

Aggregation Operations(aggregation pipeline)

Sharding

Replication

GridFS

ObjectId

Consistency

Continuing from Week3 P.60

MongoDB- Querying

find (), findOne() Operators

- find() : Returns subset of document in a collection.
  - db.collection_name.find({search_criteria}, {key_to_return : 1/0})
    - Key_to_return : 1/true- include, 0/false- exclude

```
> db.friend.find({"name":"Diane MK Woodbridge"})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howard
ts" : [ { "name" : "Abigail", "content" : "Please cook mac and cheese" }, { "name" : "Bora", "content" : "Need more
 your class." }, { "name" : "Yannet", "content" : "How's your class going?" } ] }
```

```
> db.friend.find({"name":"Diane MK Woodbridge"},{"comments":1})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "comments" : [ { "name" : "Abigail", "content" : "Please co
d", "content" : "Want to make sure you're doing fine in your class." }, { "name" : "Yannet", "content" : "H
```

MongoDB- Querying

find (), findOne() Operators

- find() : Returns subset of document in a collection.
  - db.collection_name.find({search_criteria}, {key_to_return : 1/0})
    - Key_to_return : 1- include, 0 - exclude

```
> db.friend.find({"name":"Diane MK Woodbridge"},{"comments.name":1})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "comments" : [ { "name" : "Abigail" }, { "name" : "Bora" }, { "nam
```

```
> db.friend.find({"name":"Diane MK Woodbridge"},{"comments.name":1,"_id":0})
{ "comments" : [ { "name" : "Abigail" }, { "name" : "Bora" }, { "name" : "David" }, { "name" : "Yannet" } ] }
```

```
> db.friend.find({"name":"Diane MK Woodbridge"},{"comments.name":true,"_id":false})
{ "comments" : [ { "name" : "Abigail" }, { "name" : "Bora" }, { "name" : "David" }, { "name" : "Yannet" }
```

MongoDB- Querying

Add query criteria for more complex criteria.

- Range
  - `$lt, $lte, $gt, $gte`
  - `$ne`

`db.collection_name.find({field:{range_operator:value}})`

```
> db.friend.find({"noCats": {"$gt":0}})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK
", "city" : "San Francisco", "state" : "CA" }, "noKids" : 2, "noCat
tent" : "Please cook mac and cheese" }, { "name" : "Bora", "content
ntent" : "Want to make sure you're doing fine in your class." }, {
going?" } ] }
```

```
> db.friend.find({"noKids":{"$ne":null}})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "addres
", "city" : "San Francisco", "state" : "CA" }, "noKids" : 2, "noCats" : 3, "comments" :
tent" : "Please cook mac and cheese" }, { "name" : "Bora", "content" : "Need more water
ntent" : "Want to make sure you're doing fine in your class." }, { "name" : "Yannet", "
going?" } ] }
```

https://docs.mongodb.com/v3.2/reference/operator/query/

MongoDB- Querying

Add query criteria for more complex criteria.

- OR
  - `$or` : query values for a single key.
  - `$in, $nin` : query values for multiple keys.

```
> db.friend.find({"address.city": {"$in":["San Francisco","Corte Madera"]}})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howard
", "city" : "San Francisco", "state" : "CA" }, "noKids" : 2, "noCats" : 3, "comments" : [ { "name" : "Abigail", "con
tent" : "Please cook mac and cheese" }, { "name" : "Bora", "content" : "Need more water." }, { "name" : "David", "co
ntent" : "Want to make sure you're doing fine in your class." }, { "name" : "Yannet", "content" : "How's your class
going?" } ] }
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "name" : "Yannet Interian", "address" : { "street" : "101 Howard", "
city" : "San Francisco", "state" : "CA" } }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16b"), "name" : "David Guy Brizan", "address" : { "street" : "101 Howard",
"city" : "San Francisco", "state" : "CA" } }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16c"), "name" : "David Uminsky", "address" : { "street" : "101 Howard", "ci
ty" : "San Francisco", "state" : "CA" } }
```

```
> db.friend.find({"$or":[{"address.city":{"$in": ["San Francisco","Corte Madera"]}}]})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howard
", "city" : "San Francisco", "state" : "CA" }, "noKids" : 2, "noCats" : 3, "comments" : [ { "name" : "Abigail", "con
tent" : "Please cook mac and cheese" }, { "name" : "Bora", "content" : "Need more water." }, { "name" : "David", "co
ntent" : "Want to make sure you're doing fine in your class." }, { "name" : "Yannet", "content" : "How's your class
going?" } ] }
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "name" : "Yannet Interian", "address" : { "street" : "101 Howard", "
```

https://docs.mongodb.com/v3.2/reference/operator/query/

MongoDB- Querying

Type-specific queries

- Regular Expression
  - db.collection_name.find({key:{$regex: pattern})
  - Pattern matching strings in queries
  - Follows Perl Compatible Regular Expression (PCRE) version 8.39 with UTF-8 support.

```
> db.friend.find({"name":{"$regex":'an$'}})
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "name" : "Yannet Interian", "address" : { "st
city" : "San Francisco", "state" : "CA" } }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16b"), "name" : "David Guy Brizan", "address" : { "s
"city" : "San Francisco", "state" : "CA" } }
```

https://docs.mongodb.com/manual/reference/operator/query/regex/
http://www.pcre.org/

Example

From the business collection, find all the business which “cuisine” is starting with “American”.

MongoDB- Querying

Type-specific queries

- Query Arrays
  - Same way as querying scalars.

Try it.

```
> db.food.insert({"fruit" : [ "apple", "banana", "peach", "watermelon" ]})
WriteResult({ "nInserted" : 1 })
>
> db.food.find()
{ "_id" : ObjectId("5825b56553ef451f6f65be58"), "fruit" : [ "apple", "banana", "peach", "watermelon" ] }
>
> db.food.find({"fruit":"apple"})
{ "_id" : ObjectId("5825b56553ef451f6f65be58"), "fruit" : [ "apple", "banana", "peach", "watermelon" ] }
>
```

MongoDB- Querying

Type-specific queries

- Query Arrays
  - `$slice`
    - Return a subset of elements for an array key.
    - Helpful when you know the index of the element.
    - db.collection_name.findOne(criteria, {array_name:{”$slice”: N}})
      - Number N
        - Positive N – first N elements
        - Negative N – last N elements
        - [index, # of element]

```
> db.food.find()
{ "_id" : ObjectId("58235af8f3c8812cd066eaa4"), "fruit" : [ "apple", "banana", "peach", "watermelon" ] }
```

MongoDB- Querying

Type-specific queries

- Query Arrays
  - `$slice`
    - Return a subset of elements for an array key.
    - Helpful when you know the index of the element.
    - db.collection_name.findOne(criteria, {array_name:{"$slice": N}})
      - Number N
        - Positive N – first N element
        - Negative N – last N element
        - [index, # of element]

```
> db.food.find()
{ "_id" : ObjectId("58235af8f3c8812cd066eaa4"), "fruit" : [ "apple", "banana", "peach", "watermelon" ] }
```

```
> db.food.find({"fruit":"apple"},{"fruit":{"$slice":2}})
{ "_id" : ObjectId("58235af8f3c8812cd066eaa4"), "fruit" : [ "apple", "banana" ] }
> db.food.find({"fruit":"apple"},{"fruit":{"$slice":-2}})
{ "_id" : ObjectId("58235af8f3c8812cd066eaa4"), "fruit" : [ "peach", "watermelon" ] }
> db.food.find({"fruit":"apple"},{"fruit":{"$slice":[1,2]}})
{ "_id" : ObjectId("58235af8f3c8812cd066eaa4"), "fruit" : [ "banana", "peach" ] }
```

MongoDB- Querying

Type-specific queries

- Query Arrays
  - $
    - Return the first array element matched your criteria.

Try it.

```
> db.food.find({"fruit":"apple"},{"fruit.$":1})
{ "_id" : ObjectId("58235af8f3c8812cd066eaa4"), "fruit" : [ "apple" ] }
```

MongoDB- Querying

Type-specific queries

- Query Arrays
  - $elemMatch
    - Compare all clauses in a single array element.
    - Only work for array element.

```
{ "_id" : ObjectId("58236e16f3c8812cd066eab1"), "number" : [ 5, 25 ] }
> db.favorite.find({"number":{"$gt":10, "$lt":20}})
{ "_id" : ObjectId("58236e16f3c8812cd066eab1"), "number" : [ 5, 25 ] }
> db.favorite.find({"number":{"$elemMatch":{"$gt":10, "$lt":20}}})
```

MongoDB- Querying

Type-specific queries

- Query Embedded Documents
  - Query for the entire document.
    - Find exact matches of the subdocument.
  - Query for the individual key/value pairs.
    - Using the dot notation to reach into an embedded document.

```
> db.friend.find({"address":{"street":"101 Howard"}})
```

```
> db.friend.find({"address":{"street":"101 Howard", "city":"San Francisco", "state":"CA"}})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 How
 "noCats" : 3, "comments" : [ { "name" : "Abigail", "content" : "Please cook mac and cheese" }, { "name" : "Bora"
t" : "Want to make sure you're doing fine in your class." }, { "name" : "Yannet", "content" : "How's your class g
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "name" : "Yannet Interian", "address" : { "street" : "101 Howard"
{ "_id" : ObjectId("581b610a03fdda1ba29ed16b"), "name" : "David Guy Brizan", "address" : { "street" : "101 Howard
{ "_id" : ObjectId("581b610a03fdda1ba29ed16c"), "name" : "David Uminsky", "address" : { "street" : "101 Howard",
```

```
> db.friend.find({"address.street":"101 Howard"})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howard", "cit
 "noCats" : 3, "comments" : [ { "name" : "Abigail", "content" : "Please cook mac and cheese" }, { "name" : "Bora", "content
t" : "Want to make sure you're doing fine in your class." }, { "name" : "Yannet", "content" : "How's your class going?" } ]
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "name" : "Yannet Interian", "address" : { "street" : "101 Howard", "city" :
{ "_id" : ObjectId("581b610a03fdda1ba29ed16b"), "name" : "David Guy Brizan", "address" : { "street" : "101 Howard", "city"
{ "_id" : ObjectId("581b610a03fdda1ba29ed16c"), "name" : "David Uminsky", "address" : { "street" : "101 Howard", "city" : "
```

Try it.

MongoDB- Querying

Cursors

- MongoDB returns result from find() using a cursor.
- Using cursors you can control the output of a query including skip, sort, limit, etc.

Name	Description
`cursor.addOption()`	Adds special wire protocol flags that modify the behavior of the query.'
`cursor.batchSize()`	Controls the number of documents MongoDB will return to the client in a single network message.
`cursor.close()`	Close a cursor and free associated server resources.
`cursor.comment()`	Attaches a comment to the query to allow for traceability in the logs and the system.profile collection.
`cursor.count()`	Modifies the cursor to return the number of documents in the result set rather than the documents themselves.
`cursor.explain()`	Reports on the query execution plan for a cursor.

https://docs.mongodb.com/v3.2/reference/method/js-cursor/

MongoDB- Querying

Cursors

- Using cursors you can control the output of a query including skip, sort, limit, etc.

```
> var cursor = db.friend.find().sort({"name":-1})
> cursor
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "name" : "Yannet Interian", "address" : { "street" : "101 Howard", "city" : "San Fra
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howard", "city" : "San
 "noCats" : 3, "comments" : [ { "name" : "Abigail", "content" : "Please cook mac and cheese" }, { "name" : "Bora", "content" : "Need
t" : "Want to make sure you're doing fine in your class." }, { "name" : "Yannet", "content" : "How's your class going?" } ] }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16c"), "name" : "David Uminsky", "address" : { "street" : "101 Howard", "city" : "San Franc
{ "_id" : ObjectId("581b610a03fdda1ba29ed16b"), "name" : "David Guy Brizan", "address" : { "street" : "101 Howard", "city" : "San Fr
> cursor.count()
4
```

Try
var cursor = db.friend.find()
cursor.next()

https://docs.mongodb.com/v3.0/core/cursors/

MongoDB- Indexing

- Indexing
  - B-Tree is default.
  - Optimize query performance.
    - getIndexes()
      - Check which indexes exist on a collection.
      - Default : _id (Cannot be deleted)
    - createIndex({“field” : direction})
      - Direction : 1 (Ascending), -1 (Descending)
    - dropIndex({“field” : direction})

```
db.friend.find({"name":"Yannet Interian"}).explain("executionStats")
```

Try it.

https://docs.mongodb.com/v3.0/tutorial/modify-an-index/

https://docs.mongodb.com/v3.2/tutorial/analyze-query-plan/

MongoDB- Indexing

- Indexing
  - B-Tree is default.
  - Optimize query performance.
    - getIndexes()
      - Check which indexes exist on a collection.
      - Default : _id (Cannot be deleted)
    - createIndex({“field” : direction})
      - Direction : 1 (Ascending), -1 (Descending)
    - dropIndex({“field” : direction})

Try it.

```
> db.friend.createIndex({"name":1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
db.friend.find({"name":"Yannet Interian"}).explain("executionStats")
```

https://docs.mongodb.com/v3.0/tutorial/modify-an-index/


```
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 1,
                "executionTimeMillis" : 3,
                "totalKeysExamined" : 1,
                "totalDocsExamined" : 1,
                "executionStages" : {
                        "stage" : "FETCH",
                        "nReturned" : 1,
                        "executionTimeMillisEstimate" : 0,
                        "works" : 2,
                        "advanced" : 1,
                        "needTime" : 0,
                        "needYield" : 0,
                        "saveState" : 0,
                        "restoreState" : 0,
                        "isEOF" : 1,
                        "invalidates" : 0,
                        "docsExamined" : 1,
                        "alreadyHasObj" : 0,
                        "inputStage" : {
                                "stage" : "IXSCAN",
                                "nReturned" : 1,
                                "executionTimeMillisEstimate" : 0,
                                "works" : 2,
                                "advanced" : 1,
                                "needTime" : 0,
                                "needYield" : 0,
                                "saveState" : 0,
                                "restoreState" : 0,
                                "isEOF" : 1,
                                "invalidates" : 0,
                                "keyPattern" : {
                                        "name" : 1
                                },
                                "indexName" : "name_1",
                                "isMultiKey" : false,
                                "isUnique" : false,
                                "isSparse" : false,
                                "isPartial" : false,
                                "indexVersion" : 1,
                                "direction" : "forward",
                                "indexBounds" : {
                                        "name" : [
                                                "[\"Yannet Interian\", \"Yannet Interian\"]
                                        ]
                                },
                                "keysExamined" : 1,
                                "dupsTested" : 0,
                                "dupsDropped" : 0,
```

Example

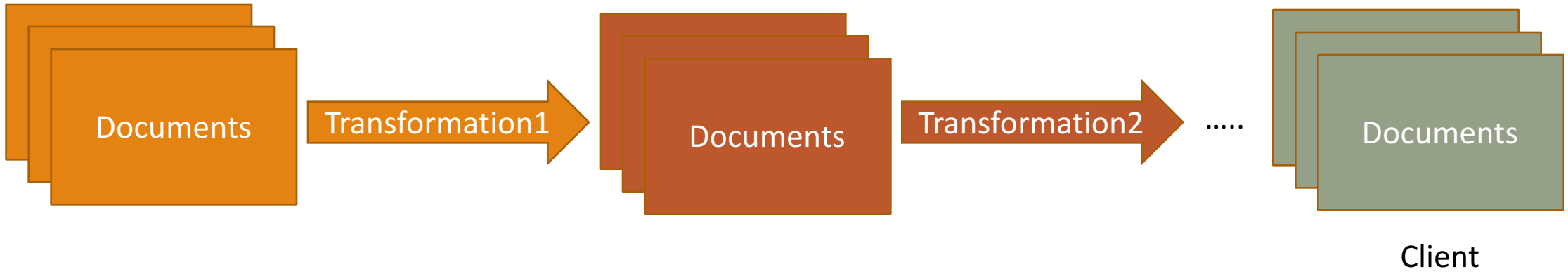
Run .dropIndex() and check which index is being utilized.

Try to .dropIndex({“_id”:1})

MongoDB

Aggregation Pipeline

- Returns an array of result document to the client. (Not write to collections.)
  - db.collecion_name.aggregate({Transform_operator_1 : criteria_1},{Transform_operator_2 : criteria_2}, …)
  - $match, $project, $lookup, $group, $unwind, $out, etc.



MongoDB

Aggregation Pipeline

- Pipeline operators
  - $match : filters document with criteria.

```
> db.friend.insert({"name": "Randy B", "address" : {"city": "Albuquerque", "state": "NM"}})
WriteResult({ "nInserted" : 1 })
> db.friend.insert({"name": "Jungin Kim", "address" : {"state" : "NC"}})
WriteResult({ "nInserted" : 1 })
```

```
> db.friend.aggregate({"$match":{"address.city":"San Francisco"}})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" :
"CA" }, "noKids" : 2, "noCats" : 3, "comments" : [ { "name" : "Abigail", "content" : "Please cook mac and
ore water." }, { "name" : "David", "content" : "Want to make sure you're doing fine in your class." }, {
oing?" } ] }
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "name" : "Yannet Interian", "address" : { "street" : "101
 } }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16b"), "name" : "David Guy Brizan", "address" : { "street" : "10
" } }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16c"), "name" : "David Uminsky", "address" : { "street" : "101 H
 }
```

MongoDB

Aggregation Pipeline

- Pipeline operators
  - $match : filters document with criteria.
  - $project : Extract field. Rename the projected field.
    - Project expressions
      - **Specify the inclusion of fields, the addition of new fields, and the resetting the values of existing fields.**
      - Mathematical - $add, $subtract, $multiply, $pow, etc.

https://docs.mongodb.com/manual/reference/operator/aggregation-arithmetic/

Name	Description
`$abs`	Returns the absolute value of a number.
`$add`	Adds numbers to return the sum, or adds numbers and a date to return a new date. If adding numbers and a date, treats the numbers as milliseconds. Accepts any number of argument expressions, but at most, one expression can resolve to a date.
`$ceil`	Returns the smallest integer greater than or equal to the specified number.
`$divide`	Returns the result of dividing the first number by the second. Accepts two argument expressions.
`$exp`	Raises *e* to the specified exponent.
`$floor`	Returns the largest integer less than or equal to the specified number.
`$ln`	Calculates the natural log of a number.
`$log`	Calculates the log of a number in the specified base.
`$log10`	Calculates the log base 10 of a number.
`$mod`	Returns the remainder of the first number divided by the second. Accepts two argument expressions.
`$multiply`	Multiplies numbers to return the product. Accepts any number of argument expressions.
`$pow`	Raises a number to the specified exponent.
`$sqrt`	Calculates the square root.
`$subtract`	Returns the result of subtracting the second value from the first. If the two values are numbers, return the difference. If the two values are dates, return the difference in milliseconds. If the two values are a date and a number in milliseconds, return the resulting date. Accepts two argument expressions. If the two values are a date and a number, specify the date argument first as it is not meaningful to subtract a date from a number.
`$trunc`	Truncates a number to its integer.

MongoDB

Aggregation Pipeline

- Pipeline operators
  - $match : filters document with criteria.
  - $project : extract field. Rename the projected field.
    - Project expressions
      - String - $concat, $substr, $toLower, $toUppper, etc.

`$concat`	Concatenates any number of strings.
`$substr`	Returns a substring of a string, starting at a specified index position up to a specified length. Accepts three expressions as arguments: the first argument must resolve to a string, and the second and third arguments must resolve to integers.
`$toLower`	Converts a string to lowercase. Accepts a single argument expression.
`$toUpper`	Converts a string to uppercase. Accepts a single argument expression.
`$strcasecmp`	Performs case-insensitive string comparison and returns: `0` if two strings are equivalent, `1` if the first string is greater than the second, and `-1` if the first string is less than the second.

https://docs.mongodb.com/manual/reference/operator/aggregation-string/

MongoDB

Aggregation Pipeline

- Pipeline operators
  - $match : filters document with criteria.
  - $project : extract field. Rename the projected field.
    - Project expressions
      - Text search, Comparison, Boolean, etc.

Try it.

```
> db.friend.aggregate({"$project":{"email":{"$toLower":{"$concat" : [{"$substr": ["$name",0,5]},"@gmail.com"]}}}})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "email" : "diane@gmail.com" }
{ "_id" : ObjectId("581ad9f603fdda1ba29ed16a"), "email" : "yanne@gmail.com" }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16b"), "email" : "david@gmail.com" }
{ "_id" : ObjectId("581b610a03fdda1ba29ed16c"), "email" : "david@gmail.com" }
{ "_id" : ObjectId("5824eafdaa0a8fda72edb9ca"), "email" : "randy@gmail.com" }
{ "_id" : ObjectId("5824eb22aa0a8fda72edb9cb"), "email" : "jungi@gmail.com" }
```

https://docs.mongodb.com/manual/reference/operator/aggregation/

MongoDB

Aggregation Pipeline

- Filter friends in Albuquerque and make their email address using their name.

```
> db.friend.aggregate({"$match":{"address.city":"Albuquerque"}},{"$project":{"email"
:{"$toLower":{"$concat" : [{"$substr": ["$name",0,5]},"@gmail.com"]}}}})
{ "_id" : ObjectId("5824eafdaa0a8fda72edb9ca"), "email" : "randy@gmail.com" }
```

MongoDB

Aggregation Pipeline

- Pipeline operators
  - $group : group documents based on certain fields and combine their values.
  - $unwind : return each of an array into a separate document.
  - $sort : collect all document and properly sort them and send the individual shard's sorted results.
  - $out : create/replace a new collection in the current database from the aggregation operation.

```
> db.friend.aggregate({"$unwind":"$comments"},{"$sort":{"comments.name":-1}})
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howa
mments" : { "name" : "Yannet", "content" : "How's your class going?" } }
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howa
mments" : { "name" : "David", "content" : "Want to make sure you're doing fine in your class." } }
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howa
mments" : { "name" : "Bora", "content" : "Need more water." } }
{ "_id" : ObjectId("581ad92303fdda1ba29ed169"), "name" : "Diane MK Woodbridge", "address" : { "street" : "101 Howa
mments" : { "name" : "Abigail", "content" : "Please cook mac and cheese" } }
```

Example

Using aggregate operators,

- Filter friend living in “101 Howard”, “San Francisco”
- Make their email address being first 4 letters of their name with @usfca.edu.

HW 2

Due by November 17th (Midnight).

Why are we doing this?!

If you are collaborating,

- List your collators and contributions of each member in Collaborators.txt.
- You can work up to 3 people together.
- At least change the variable names. (This collaboration doesn't mean it is a group project. Understand and write your own code.)

References

Chodorow, Kristina. *MongoDB: the definitive guide*. O'Reilly Media, Inc., 2013.

MongoDB. *MongoDB Documentation*, https://docs.mongodb.com/, 2016.