

# Tasks

## Contents

<b>Task 1.....</b>	<b>1</b>
<b>Business Documentation .....</b>	<b>1</b>
User .....	1
Rules .....	2
Tasks .....	2
<b>Task 2.....</b>	<b>5</b>
<b>Task 3.....</b>	<b>6</b>
<b>Task 4.....</b>	<b>11</b>
<b>Task 5.....</b>	<b>13</b>
<b>Task 6.....</b>	<b>20</b>

## Task 1

Provide the problem requirements and the conceptual model in UML for your project. You can reuse the ones made in Project 1.

## Business Documentation

### User

#### Airbnb Hosts

Airbnb Host can use this database and management application to manage their profile and check their ratings.

#### Airbnb Staff

Airbnb staff can use this database and application check the status, profile and rating of Hosts. They can also update the rating of Hosts.

## Rules

This database aim to be an internal tool to manage Airbnb Host.

- As for the Data schema, I used the data from Airbnb's website: <http://insideairbnb.com/get-the-data.html>.
- This internal host management tool is used for Airbnb hosts and Airbnb staff to manage the hosts' profile, listing information and listing reviews.
- New airbnb hosts can create their account in this application. In the right side of the website, there is a form to submit their name and email to create a new account. Besides, they will be given a new and unique *\_id* by MongoDB, which is hidden in this page but will be recorded in the database.
- A new Airbnb host has the default rating as 5 starts.
- Airbnb hosts can update their information in this application, for example, update their name and email address
- Airbnb staff do not have the access to update the rating of hosts based on their listings rating. Because the *hostRating* which is a derived attribute, is calculated by their listings' s ratings.
- The *hostRating* attribute does not physical exist in our database, and it is calculated base the rating attribute in *listings* collection.
- The *guests* collection stores the guests' data of Airbnb.
- Once guests have stayed in one listing, their record will be paired.

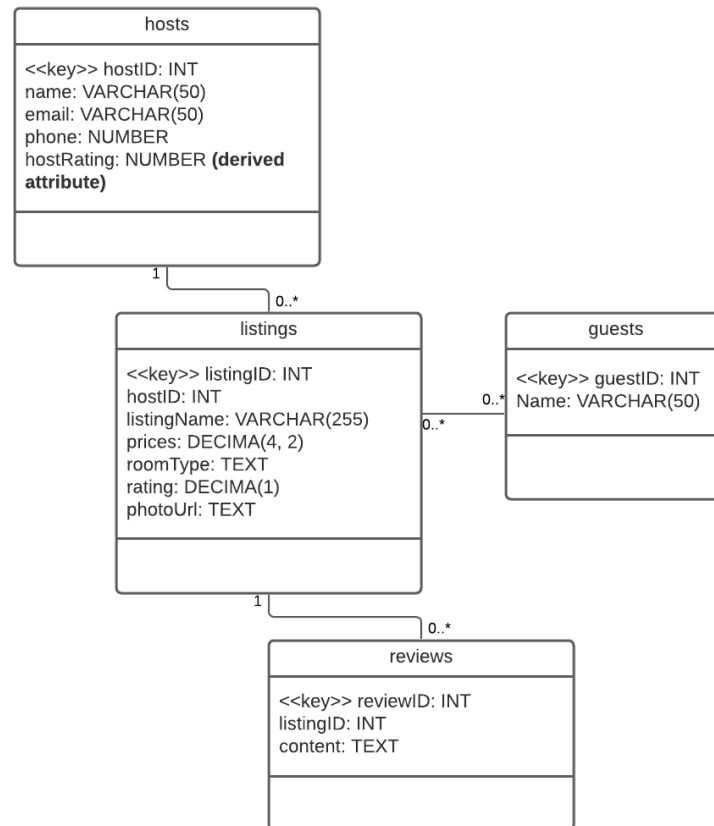
## Tasks

The above rules require us to create four tables and four routes in UML. I used LucidChart to create UML for these tables,

- The first table is *hosts* collection.
- The second table is *listings* collection.
- The third table is *reviews* collection.
- The fourth table is *guests* collection.

## Airbnb Hosts Management UML

Ying Tuo



- Our database should build appropriate schema for this database, so that each tables can be associate with others and build relationship.
- In the listings table, **hostID** is the foreign key for this listing. (**Note**, the foreign should be the `_id` that automatically generate by MongoDB, but it is hard to generate `_id` with the same regular expression test data, we use hosted here to denote the foreign keys and build relationships.)
- In order to develop this application, helping Airbnb and Airbnb Host to manage their data, we should have the following APIs to operate our database.
- `/hosts` The home page will redirect to (`/hosts`), which is the main routes. In this page, it shows the Airbnb host database with their name, email, the date they joined Airbnb and most importantly, their `hostRating`.
- `/hosts/create` The CREATE interface let users to create a new Airbnb user, and post data to database.
- `/hosts/delete` This route let us to delete the records in the Hosts table. Besides, because the `hostid` in the Hosts table is foreign key in Listings table, and `listingid` in Listing Table is the foreign key in Reviews Table, all of the related records will be deleted.
- `/hosts/update` This route let us to update the information and records in the Hosts table.

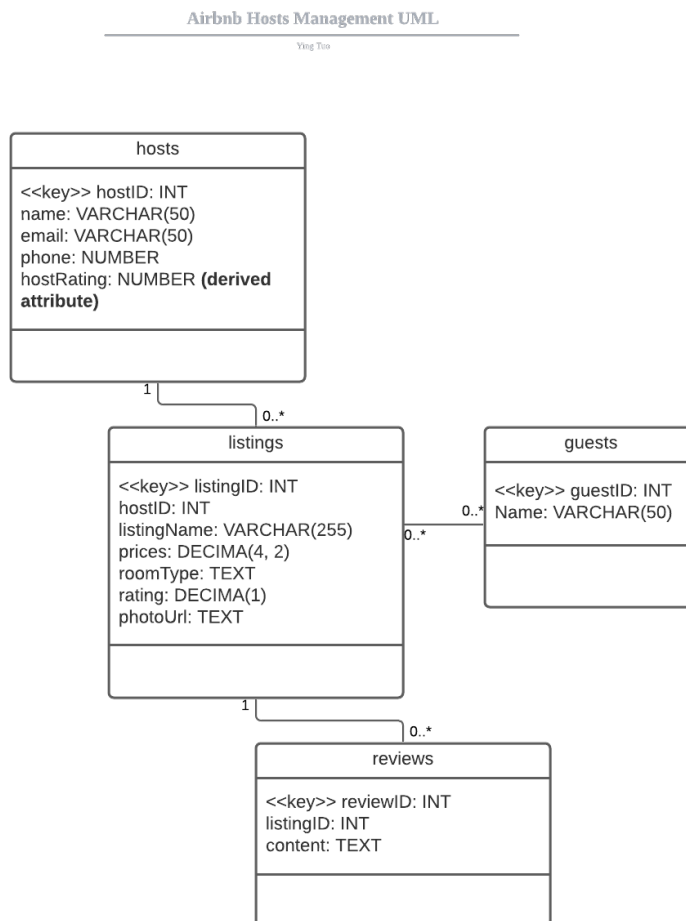


## Task 2

Adapt the Logical Data model from your Project 2 to have hierarchical tables. This is, main (root) tables from which all the other tables relate to. This main tables will become later your Mongo Collections. From your main tables you can have aggregation/composition, one to many and many to many relationships.

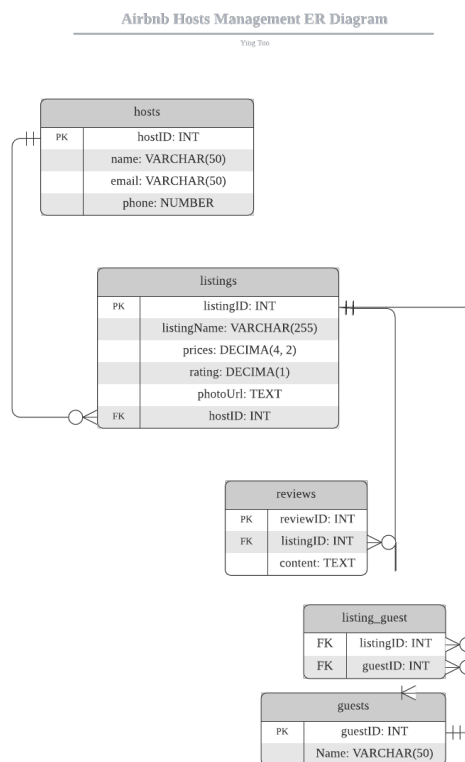
I used LucidChart to build my UML and ERD. In the UML diagram, there are four classes, which are four tables, **hosts**, **listings**, **guests**, and **reviews**. The relationship between listings and reviews is one to many; the relationship between hosts and listings is one to one; the relationship between listings and guests is many to many.

The description of these attributes and schema are below, and I build a derived attribute, **hostRating**, which is calculated based on the **rating** attribute in the Listings table. This derived attribute does not physically exist in our database, but it can be derived from other tables and shows in our Web UI.



The ER Diagram. As the description in the above question, the relationship between hosts to listings is **zero to many**, while the relationship between listings to hosts is **one to many**; The relationship between listings to reviews is **one to many**, while the relationship between reviews to listings is **one to one**; the relationship between listings to guests is **many to many**.

In order to hold the relationship of many-to-many relationship between listings to guests in ERD, we should build an association entities collection, listing\_guest.



Therefore, my ERD will finally have five tables.

### Task 3

From this logical model define the main Collections (Documents/Tables) you will be using in your Mongo Database. Provide a couple of JSON examples of these objects with comments when necessary. Think about a document that you will give to another database engineer that would take over your database.

The collections schema looks like below,

#### hosts Collection

```
{
  "_id": "ObjectID",
  "hostid": "Int32",
  "name": "String",
  "email": "String",
  "phone": "String"
  "hostsListings":{
    "_id": "ObjectID",
    "listingid": "Int32",
    "hostid": "Int32",
    "name": "String",
    "price": "Int32",
    "rating": "Int32",
    "photoUrl": "String"
  }
}
```

### listings Collection

```
{
  "_id": "ObjectID",
  "listingid": "Int32",
  "hostid": "Int32",
  "name": "String",
  "price": "Int32",
  "rating": "Int32",
  "photoUrl": "String"
}
```

### reviews Collection

```
{
  "reviewid": "Int32",
  "listingid": "Int32",
  "content": "String",
}
```

### guests Collection

```
{
  "guestid": "Int32",
  "name": "String",
}
```

## listing\_guest Collection

```
{
  "listingid": "Int32",
  "hostid": "Int32",
}
```

My main collection is hosts, and the Schema in MongoDB looks like below,

```
{
  "fields": [
    {
      "name": "_id",
      "path": "_id",
      "count": 200,
      "types": [
        {
          "name": "ObjectID",
          "bsonType": "ObjectID",
          "path": "_id",
          "count": 200,
          "values": [
            "5fbb3903b032ee8cebfb5fb3",
            "5fbb3903b032ee8cebfb5fb4",
            "5fbb3903b032ee8cebfb5fb5",
            ...
          ],
          "total_count": 0,
          "probability": 1,
          "unique": 200,
          "has_duplicates": false
        }
      ],
      "total_count": 200,
      "type": "ObjectID",
      "has_duplicates": false,
      "probability": 1
    },
    {
      "name": "email",
      "path": "email",
      "count": 200,
```



```
"types": [
  {
    "name": "String",
    "bsonType": "String",
    "path": "email",
    "count": 200,
    "values": [
      "geastope0@behance.net",
      "ghebbard2@amazon.com",
      "arobken3@soup.io",
      ...
    ],
    "total_count": 0,
    "probability": 1,
    "unique": 100,
    "has_duplicates": false
  }
],
"total_count": 200,
"type": "String",
"has_duplicates": false,
"probability": 1
},
{
  "name": "hostid",
  "path": "hostid",
  "count": 200,
  "types": [
    {
      "name": "Int32",
      "bsonType": "Int32",
      "path": "hostid",
      "count": 200,
      "values": [
        1,
        2,
        3,
        ...
      ],
    },
    "total_count": 0,
    "probability": 1,
```

```
        "unique": 200,
        "has_duplicates": false
    }
],
"total_count": 200,
"type": "Int32",
"has_duplicates": false,
"probability": 1
},
{
    "name": "name",
    "path": "name",
    "count": 200,
    "types": [
        {
            "name": "String",
            "bsonType": "String",
            "path": "name",
            "count": 200,
            "values": [
                "Gaile Hebbard",
                "Alejandro Robken",
                "Lief Burfield",
                ...
            ],
            "total_count": 0,
            "probability": 1,
            "unique": 100,
            "has_duplicates": false
        }
    ],
    "total_count": 200,
    "type": "String",
    "has_duplicates": false,
    "probability": 1
}
],
"count": 200
}
```

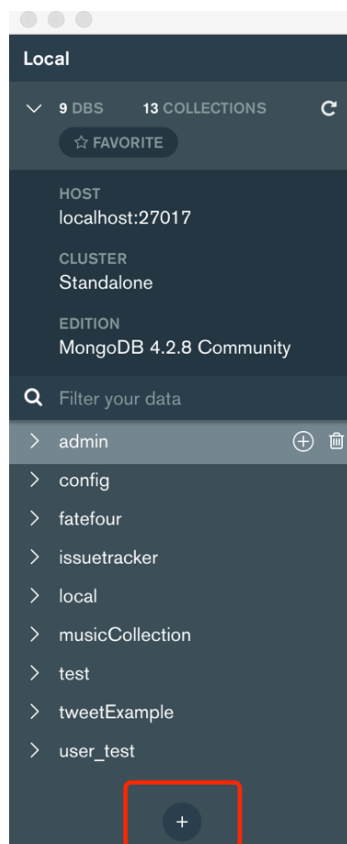
## Task 4

Populate the tables with test data. You can use tools such as <https://www.mockaroo.com/schemas> or <https://www.generatedata.com>. You can export the sample data to CSV and then use mongoimport or Mongo Compass to populate your tables. Include in your repository a dump file that can be used to regenerate your database, and the instructions on how to initialize it

I have uploaded the JSON files in the db\_Data folder, and we should use MongoDB compass to import these data,



1. Firstly, building a new database in your local MongoDB database,



### Create Database

Database Name

airbnb

Collection Name

hosts

☐ Capped Collection ⓘ

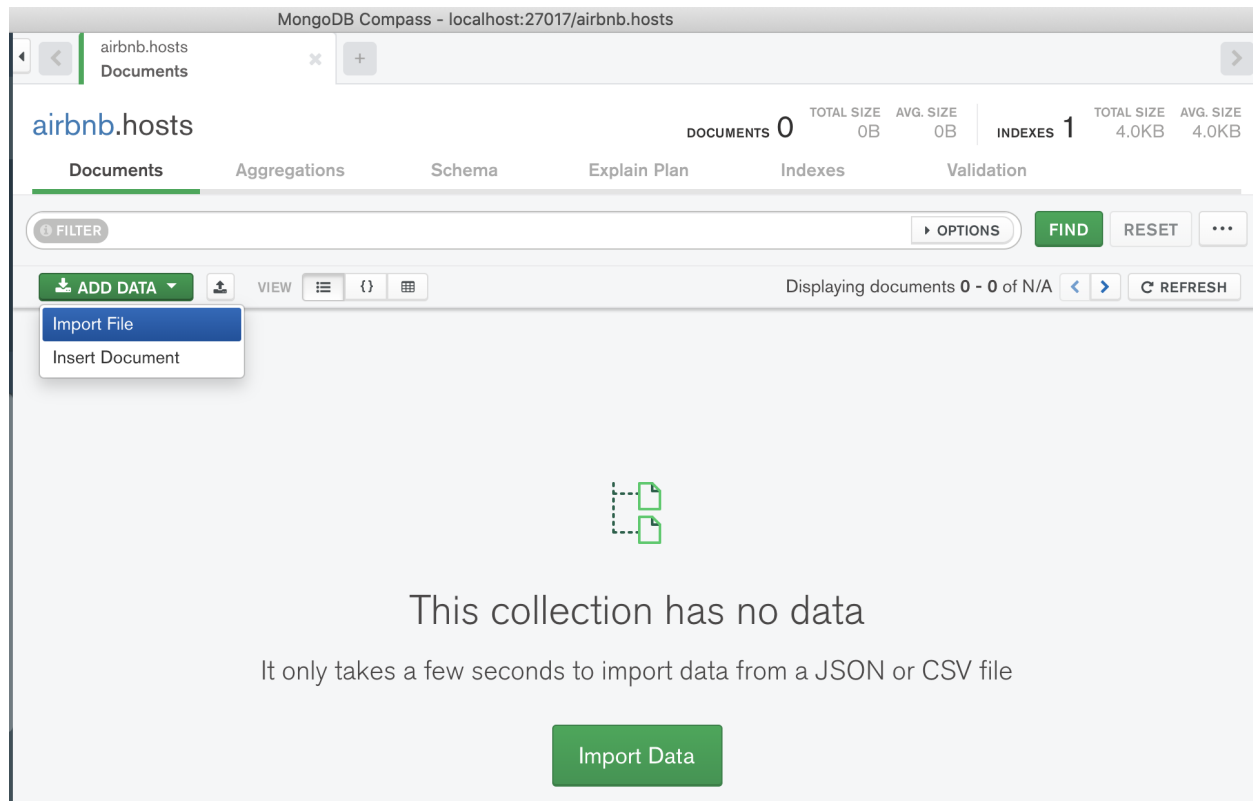
☐ Use Custom Collation ⓘ

Before MongoDB can save your new database, a collection name must also be specified at the time of creation. [More Information](#)

CANCEL

CREATE DATABASE

2. After building these five collections, we can import JSON files into these collections,



Import To Collection airbnb.hosts

### Select File

/Users/cortey/Documents/GitHub/airbnbHost\_MongoDB/db\_Data/Hosts.js

BROWSE

### Select Input File Type

JSON

CSV

### Options

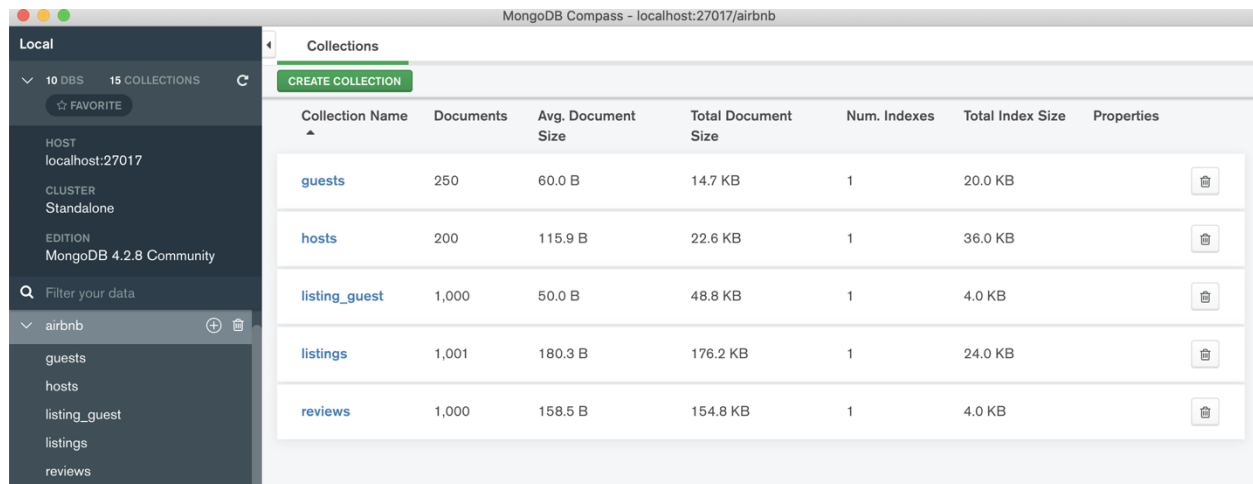
☒ Ignore empty strings

☐ Stop on errors

CANCEL

IMPORT

- Finally, all of the collections has been built, and we can start to test our data and applicatio



MongoDB Compass - localhost:27017/airbnb

Local

10 DBS 15 COLLECTIONS

☆ FAVORITE

HOST  
localhost:27017

CLUSTER  
Standalone

EDITION  
MongoDB 4.2.8 Community

Filter your data

airbnb

collections

CREATE COLLECTION

Collection Name	Documents	Avg. Document Size	Total Document Size	Num. Indexes	Total Index Size	Properties
guests	250	60.0 B	14.7 KB	1	20.0 KB	
hosts	200	115.9 B	22.6 KB	1	36.0 KB	
listing_guest	1,000	50.0 B	48.8 KB	1	4.0 KB	
listings	1,001	180.3 B	176.2 KB	1	24.0 KB	
reviews	1,000	158.5 B	154.8 KB	1	4.0 KB	

## Task 5

Define and execute at least five queries that show your database. At least one query must contain and aggregation <https://docs.mongodb.com/manual/aggregation/>, one must contain a complex search criterion (more than one expression with logical connectors), one should be counting documents for a specific user, and one must be updating a document based on a query parameter (e.g. flipping on or off a boolean attribute for a document, such as enabling/disabling a song)

### 1. One query must contain and **aggregation**.

This MongoDB query aggregate two collections, hosts and listings, so that we can have an Embedded JSON Document.

```
const query = [
  {
    '$lookup': {
      'from': 'listings',
      'localField': 'hostid',
      'foreignField': 'hostid',
      'as': 'hostsListings'
    }
  }
]

airbnb.hosts.aggregate(query);
```

airbnb.hosts

DOCUMENTS 200 TOTAL SIZE 22.6KB AVG. SIZE 116B INDEXES 1 TOTAL SIZE 36.0KB AVG. SIZE 36.0KB

Documents Aggregations Schema Explain Plan Indexes Validation

COLLATION read-Modified SAVE SAMPLE MODE AUTO PREVIEW

Output after \$lookup stage (Sample of 20 documents)

```

1 // **
2 * from: The target collection.
3 * localField: The local join field.
4 * foreignField: The target join field.
5 * as: The name for the results.
6 * pipeline: The pipeline to run on the joined collection.
7 * let: Optional variables to use in the pipeline.
8 */
9 {
10   from: 'listings',
11   localField: 'hostid',
12   foreignField: 'hostid',
13   as: 'hostsListings'
14 }

```

Output after \$lookup stage (Sample of 20 documents)

```

{
  "_id": ObjectId("5fbb3903b032ee8cebfb5fb3"),
  "hostid": 1,
  "name": "Garrek Eastope",
  "email": "geastope@behance.net",
  "phone": "901-103-2214",
  "hostsListings": Array
    0: Object
    1: Object
    2: Object
    3: Object
    4: Object
}

```

```

{
  "_id": ObjectId("5fbb3903b032ee8cebfb5fb4"),
  "hostid": 2,
  "name": "Dicky Braemer",
  "email": "dbraemer1@netscape.com",
  "phone": "618-707-3055",
  "hostsListings": Array
    0: Object
    1: Object
    2: Object
    3: Object
    4: Object
}

```

- By using the \$group in MongoDB Aggregations, we can get the derived attribute, hostRating from the \$avg of listings.rating, which is a cross collections query. And this output is what I will show in the Project UI.

```

const query = [
  {
    "$lookup": {
      "from": "listings",
      "localField": "hostid",
      "foreignField": "hostid",
      "as": "hostsListings"
    }
  }, {
    "$unwind": "$hostsListings"
  }, {
    "$unwind": "$_id"
  }, {
    "$group": {
      "_id": "$_id",
      "hostid": {
        "$first": "$hostid"
      },
      "name": {
        "$first": "$name"
      },
      "email": {
        "$first": "$email"
      },
      "phone": {

```

```

        "$first": "$phone"
      },
      "hostRating": {
        "$avg": "$hostsListings.rating"
      }
    }
  }
];
airbnb.hosts.aggregate(query);

```

airbnb.hosts

Documents Aggregations Schema Explain Plan Indexes Validation

COLLATION read- Modified SAVE

Output after \$unwind stage (Sample of 20 documents)

```

1 /**
2  * path: Path to the array field.
3  * includeArrayIndex: Optional name for index.
4  * preserveNullAndEmptyArrays: Optional
5  * toggle to unwind null and empty values.
6  */
7 "$hostsListings"
8

```

```

_id: ObjectId("5fbb3903b032ee8cebfb5fb3")
hostid: 1
name: "Garrek Eastope"
email: "geastope@behance.net"
phone: "901-103-2214"
hostsListings: Object

```

```

_id: ObjectId("5fbb3903b032ee8cebfb5fb3")
hostid: 1
name: "Garrek Eastope"
email: "geastope@behance.net"
phone: "901-103-2214"
hostsListings: Object
  _id: ObjectId("5fbb39a7b032ee8cebfb6119")
  listingid: 149
  hostid: 1

```

Output after \$unwind stage (Sample of 20 documents)

```

1 /**
2  * path: Path to the array field.
3  * includeArrayIndex: Optional name for index.
4  * preserveNullAndEmptyArrays: Optional
5  * toggle to unwind null and empty values.
6  */
7 "$_id"

```

```

_id: ObjectId("5fbb3903b032ee8cebfb5fb3")
hostid: 1
name: "Garrek Eastope"
email: "geastope@behance.net"
phone: "901-103-2214"
hostsListings: Object

```

```

_id: ObjectId("5fbb3903b032ee8cebfb5fb3")
hostid: 1
name: "Garrek Eastope"
email: "geastope@behance.net"
phone: "901-103-2214"
hostsListings: Object

```

Output after \$group stage (Sample of 20 documents)

```

1 /**
2  * _id: The id of the group.
3  * fieldN: The first field name.
4  */
5 {
6   _id: '$_id',
7   hostid: {'$first: '$hostid'},
8   name: {'$first: '$name'},
9   email: {'$first: '$email'},
10  phone: {'$first: '$phone'},
11  hostRating: {
12    $avg: '$hostsListings.rating',
13  }
14 }
15 //name: {$trim: '$email'},
16 }

```

```

_id: ObjectId("5fbb3903b032ee8cebfb6024")
hostid: 114
name: "Renado Franek"
email: "rfranek35@google.com.br"
phone: "858-956-1114"
hostRating: 2.75

```

```

_id: ObjectId("5fbb3903b032ee8cebfb6028")
hostid: 118
name: "Clotilda Hintze"
email: "chintze39@joomla.org"
phone: "703-455-1632"
hostRating: 3

```

- One must contain a complex search criterion (more than one expression with logical connectors)

Base on the output from the former query, we can filter the users whose hostRating is larger than 2.

```
const query = [
  {
    '$lookup': {
      'from': 'listings',
      'localField': 'hostid',
      'foreignField': 'hostid',
      'as': 'hostsListings'
    }
  }, {
    '$unwind': '$hostsListings'
  }, {
    '$unwind': '$_id'
  }, {
    '$group': {
      '_id': '$_id',
      'hostid': {
        '$first': '$hostid'
      },
      'name': {
        '$first': '$name'
      },
      'email': {
        '$first': '$email'
      },
      'phone': {
        '$first': '$phone'
      },
      'hostRating': {
        '$avg': '$hostsListings.rating'
      }
    }
  }, {
    '$project': {
      '_id': 1,
      'hostid': 1,
      'name': 1,
      'email': 1,
      'phone': 1,
      'hostRatingGte2': {
        '$gte': [
          '$hostRating', 2
        ]
      }
    }
  }
]
```



```

    ]
  }
}
}
]
airbnb.hosts.aggregate(query);

```

The screenshot shows the MongoDB Compass interface for the 'airbnb.hosts' collection. The 'Aggregations' tab is selected, displaying two stages: '\$group' and '\$project'.

**\$group stage:** The output shows documents grouped by 'hostid'. For example, hostid 114 has a 'hostRating' of 2.75, and hostid 118 has a 'hostRating' of 3.

**\$project stage:** The output shows documents with fields projected. For example, hostid 114 has 'hostRatingGte2: true'.

#### 4. One should be counting documents for a specific user

I use counting documents method to get the total number of listings of one user.

```

const query = [
  {
    '$lookup': {
      'from': 'hosts',
      'localField': 'hostid',
      'foreignField': 'hostid',
      'as': 'hostsListings'
    }
  }, {
    '$unwind': '$hostsListings'
  }
];

```

```

    }, {
      '$group': {
        '_id': '$hostsListings._id',
        'count': {
          '$sum': 1
        }
      }
    }
  ]
}

airbnb.listings.aggregate(query);

```

airbnb.listings

Documents Aggregations Schema Explain Plan Indexes Validation

COLLATION count- Modified SAVE

**\$lookup** Output after \$lookup stage (Sample of 20 documents)

```

1 /**
2  * from: The target collection.
3  * localField: The local join field.
4  * foreignField: The target join field.
5  * pipeline: The pipeline to run on the joined collection.
6  * as: The name for the results.
7  * let: Optional variables to use in the pipeline.
8  */
9 {
10  from: 'hosts',
11  localField: 'hostid',
12  foreignField: 'hostid',
13  as: 'hostsListings'
14 }

```

Output after \$lookup stage (Sample of 20 documents)

```

{
  "_id": "ObjectId('5fbb39a7b032ee8cebfb6086')",
  "listingid": 2,
  "hostid": 77,
  "name": "Heuchera parvifolia Nutt. ex Torr. & A. Gra...",
  "price": 215,
  "rating": 1,
  "photoUrl": "http://dummyimage.com/245x200.bmp/5fa2d...",
  "hostsListings": [
    {
      "_id": "ObjectId('5fbb3903b032ee8cebfb5fff')",
      "hostid": 77,
      "name": "Sandve Bedlington"
    }
  ]
}

```

**\$unwind** Output after \$unwind stage (Sample of 20 documents)

```

1 /**
2  * path: Path to the array field.
3  * includeArrayIndex: Optional name for index.
4  * preserveNullAndEmptyArrays: Optional
5  * toggle to unwind null and empty values.
6  */
7 { "hostsListings" }

```

Output after \$unwind stage (Sample of 20 documents)

```

{
  "_id": "ObjectId('5fbb39a7b032ee8cebfb6086')",
  "listingid": 2,
  "hostid": 77,
  "name": "Heuchera parvifolia Nutt. ex Torr. & A. Gra...",
  "price": 215,
  "rating": 1,
  "photoUrl": "http://dummyimage.com/245x200.bmp/5fa2d...",
  "hostsListings": {
    "_id": "ObjectId('5fbb3903b032ee8cebfb5fff')",
    "hostid": 77,
    "name": "Sandve Bedlington"
  }
}

```

**\$group** Output after \$group stage (Sample of 20 documents)

```

1 /**
2  * _id: The id of the group.
3  * fieldN: The first field name.
4  */
5 {
6  _id: "$hostsListings._id",
7  count: { $sum: 1 }
8 }

```

Output after \$group stage (Sample of 20 documents)

```

{
  "_id": "ObjectId('5fbb3903b032ee8cebfb605d')",
  "count": 4
}

```

**\$group** Output after \$group stage (Sample of 20 documents)

```

{
  "_id": "ObjectId('5fbb3903b032ee8cebfb6061')",
  "count": 6
}

```

5. One must be updating a document based on a query parameter

I use this MongoDB query to do Update operation in my Node + Express application

```

airbnb.hosts.updateOne(
  { _id: ObjectId(host._id) },

```

```

{
  $set: {
    name: host.name,
    email: host.email,
    phone: host.phone
  },
}
}
);

```

→ ↺ ⓘ localhost:3000/hosts

# Hosts

## Current Hosts

Name: Email: Phone:

Ying Tuo	tuo.y@northeastern.edu	347-651-8167
----------	------------------------	--------------

🌟Rating🌟: 5

X Update

```

86 myDB.updateHost = async function (host) {
87   const client = MongoClient(uri, { useUnifiedTopology: true });
88   try {
89     await client.connect();
90     const db = client.db(dbName);
91     const col = db.collection(colName);
92     console.log(host._id);
93     console.log(host.name);
94     console.log(host.email);
95     console.log(host.phone);
96   } catch (err) {
97     console.error(err);
98   }
99 }

```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE 2: node

```

POST /hosts/update 302 60.993 ms - 56
GET /hosts 200 498.458 ms - 23196
GET /stylesheets/style.css 304 1.436 ms - --
(nodeemon) restarting due to changes...
(nodeemon) starting node ./bin/www
(nodeemon) restarting due to changes...
(nodeemon) starting node ./bin/www
express-session deprecated undefined resave option; provide resave option app.js:14:3
express-session deprecated undefined saveUninitialized option; provide saveUninitialized option app.js:14:5
5fbb45aa64cd1dabe870ebd5
Ying T
tuo.y@northeastern.edu
3476518167
update CommandResult {
  result: { n: 1, nModified: 1, ok: 1 },
}

```

## Hosts

Host Updated

## Current Hosts

Name: Email: Phone:

Ying T	tuo.y@northeastern.edu	3476518167
--------	------------------------	------------

🌟Rating🌟: 5

X Update

**Task 6**

Create a basic Node + Express application that let's you create, display, modify and delete at least two of the tables. One of the tables can be the users table. No need to have a polished interface, and you can use the code created in class as a starting point, and/or the code you created for Project 1

My repo for this project is: [https://github.com/tuoying96/airbnbHost\\_MongoDB](https://github.com/tuoying96/airbnbHost_MongoDB)