Tasks

Contents

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

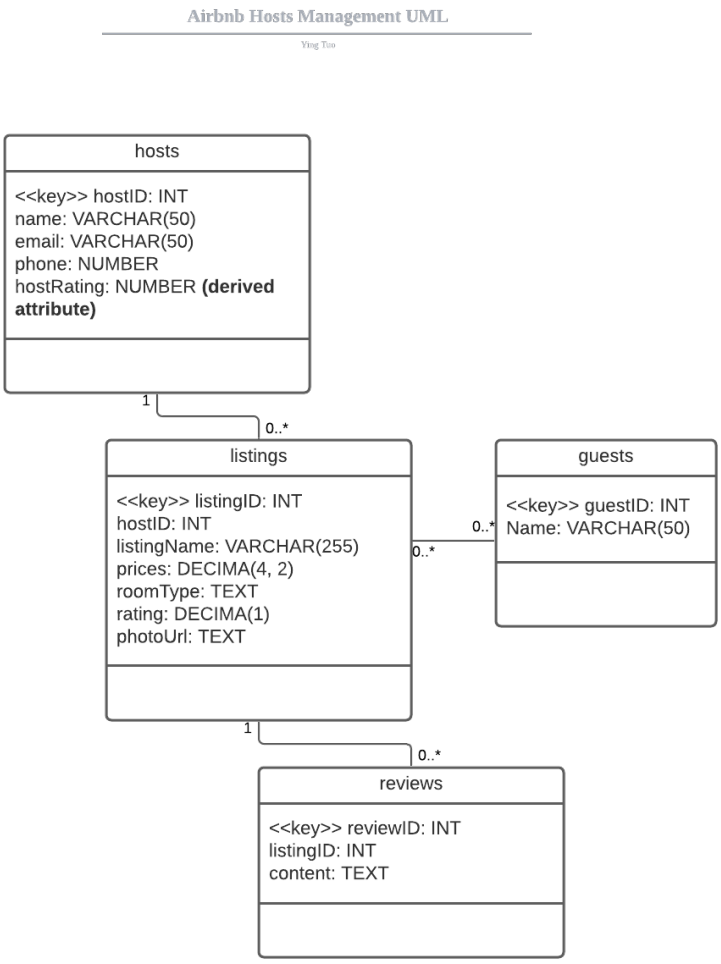
Airbnb staff can use this database and application check the status, profile and rating of Hosts. They can also update the rating of Hosts.

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. Bresides, 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.

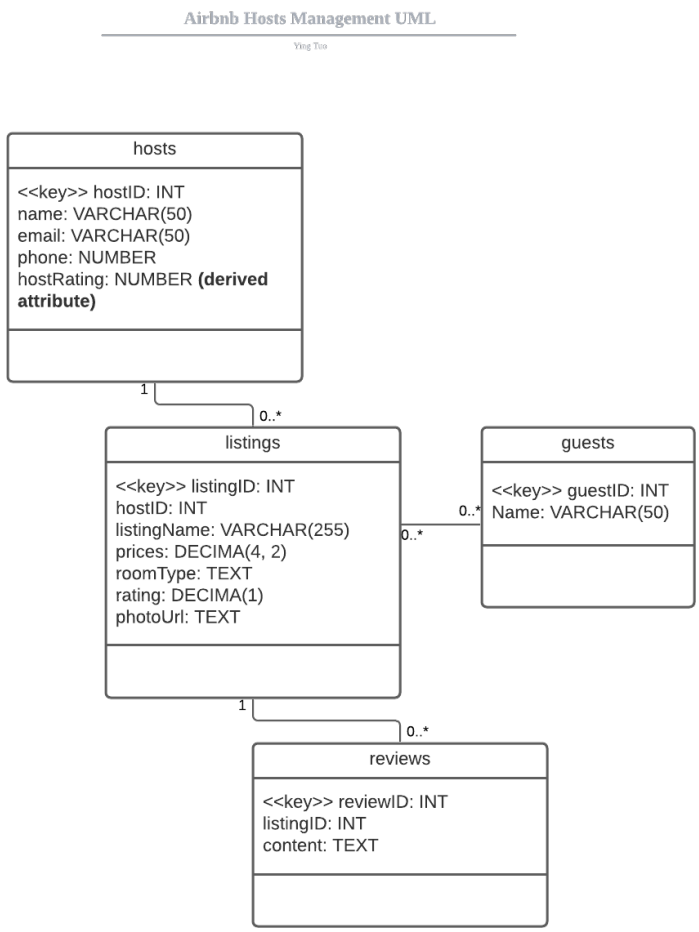


- Our database should build appropriate schema for this database, so that each tables can be associate with others and build relationship.
- In the listings table. **hostID** is the foreign key for this listing. (**Note**, the foreign should be the _id that automatically generate by MongDB, but it is 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 tht 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 bbe deleted.
- `/hosts/update` This route let us to uodate the information and records in the Hosts table.

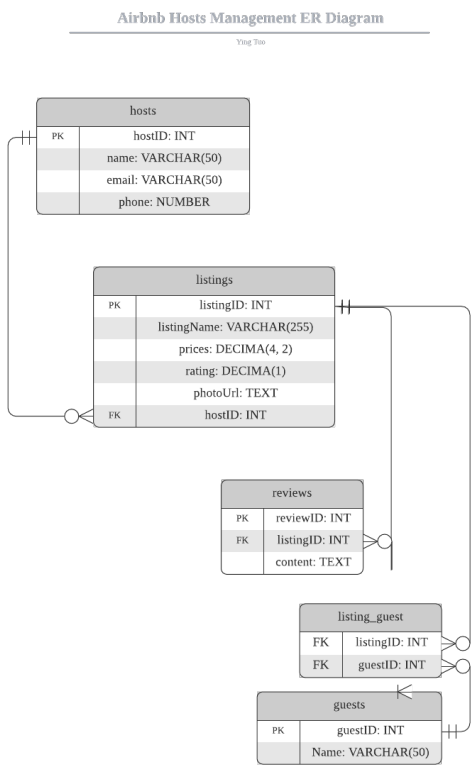
Task 2

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,
```

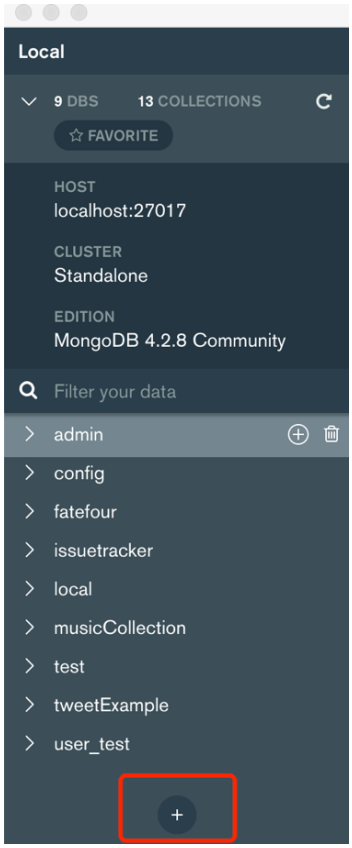
```
          "unique": 200,
          "has_duplicates": false
        }
      ],
      "total_count": 200,
      "type": "Int32",
      "has_duplicates": false,
      "probability": 1
    },
    {
      "name": "name",
      "path": "name",
      "count": 200,
      "types": [
        {
          "name": "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
}
```

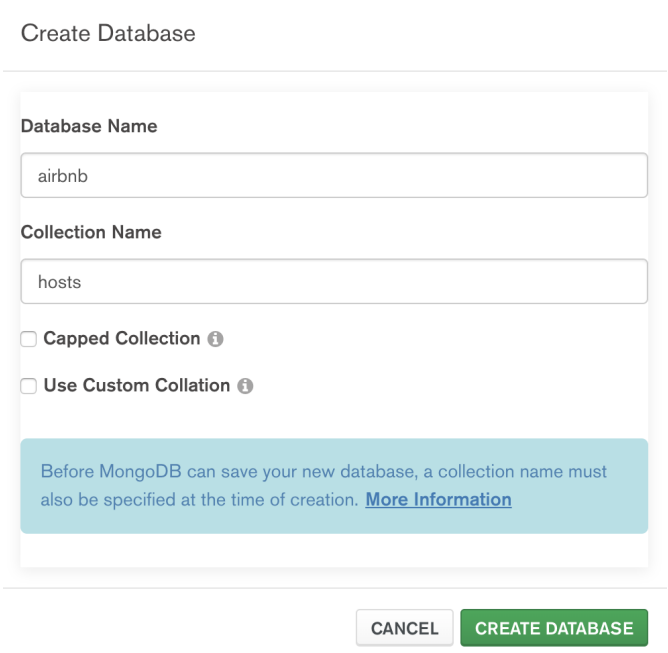
Task 4

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

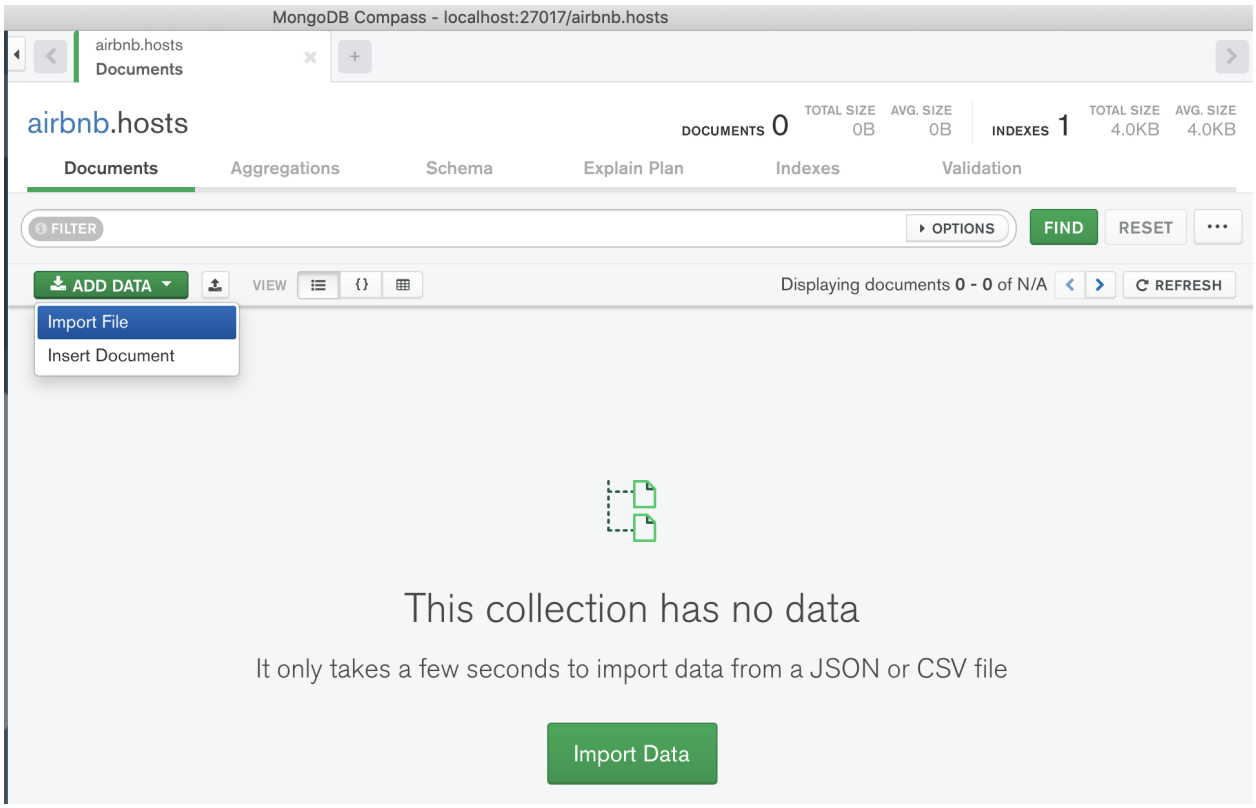
I have uploaded the JSON files in the db_Data folder, and we should use MongoDB compass to import these data,

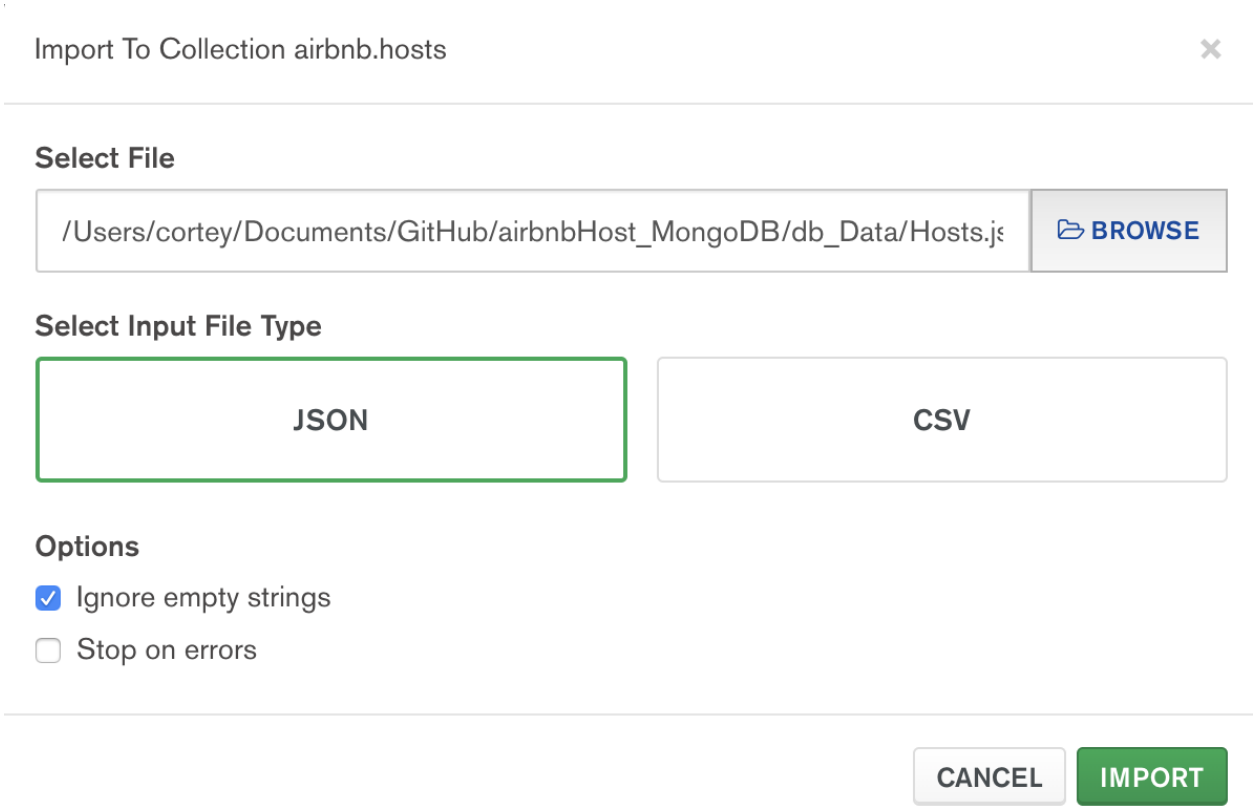
1. Firstly, building a new database in your local MongoDB database,



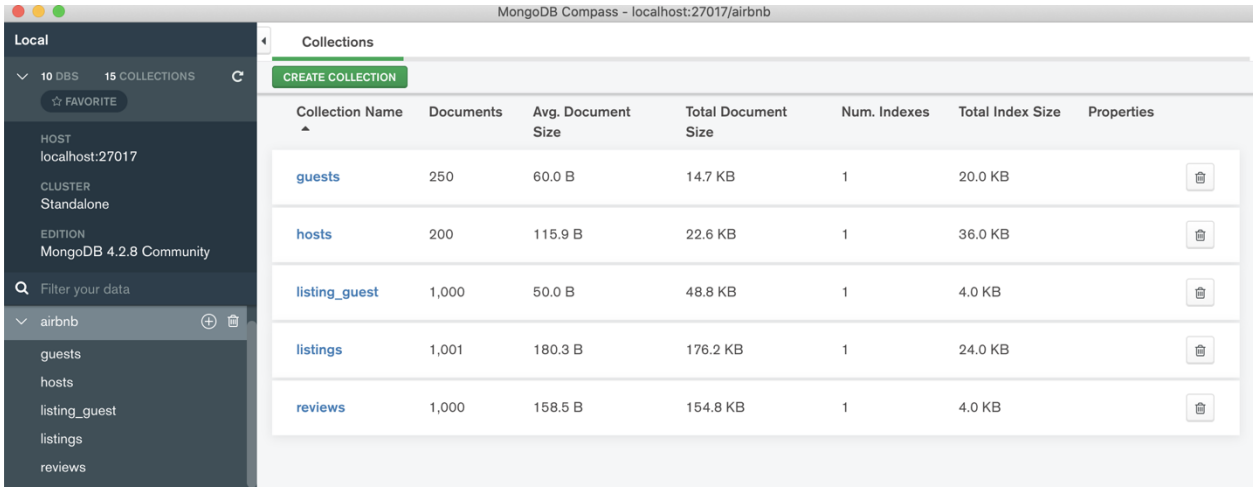


2. After building these five collections, we can import JSON files into these collections,





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



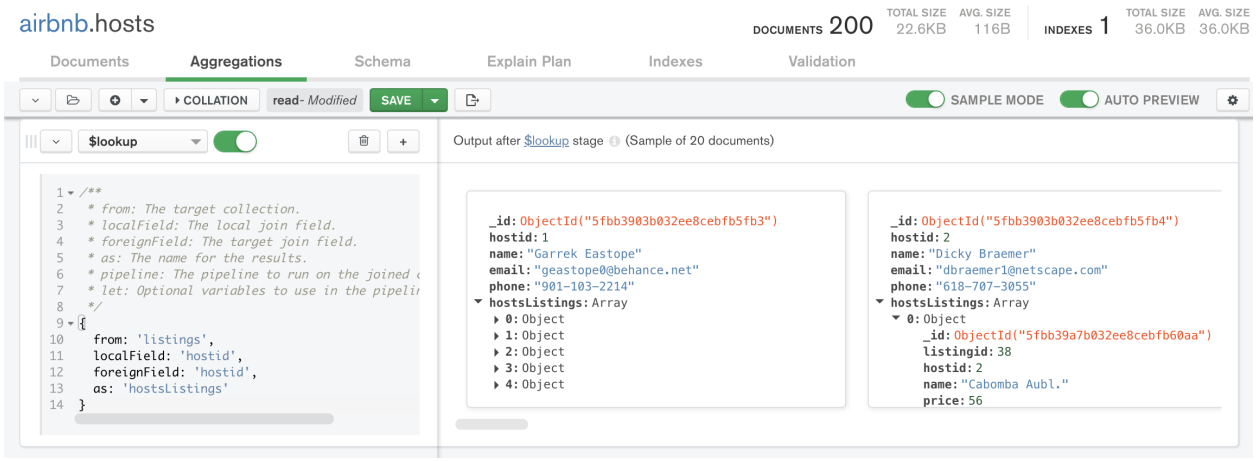
Task 5

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.

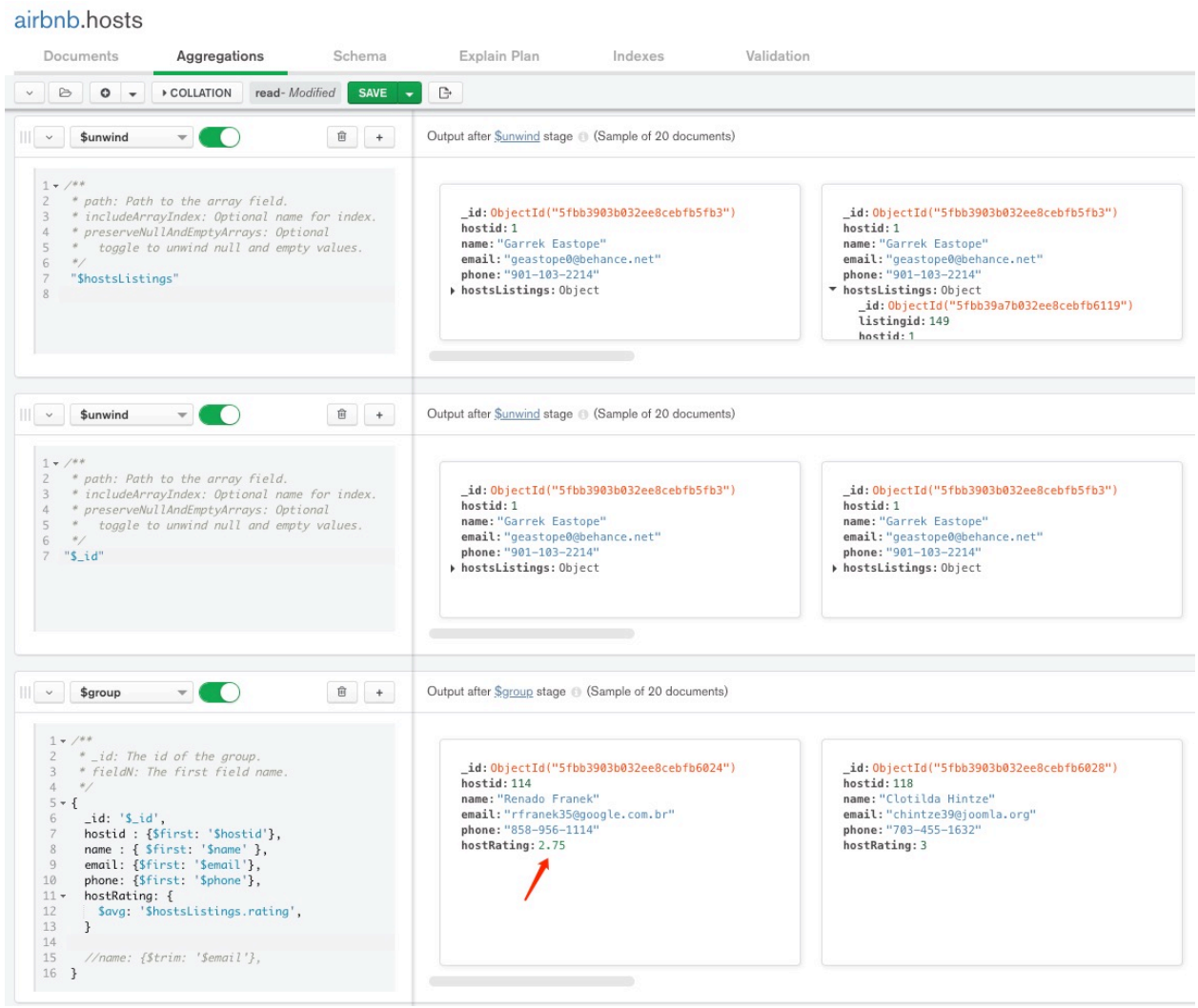
```
const query = [
        {
          '$lookup': {
            'from': 'listings',
            'localField': 'hostid',
            'foreignField': 'hostid',
            'as': 'hostsListings'
          }
        }
      ]
airbnb.hosts.aggregate(query);
```



2. 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);
```

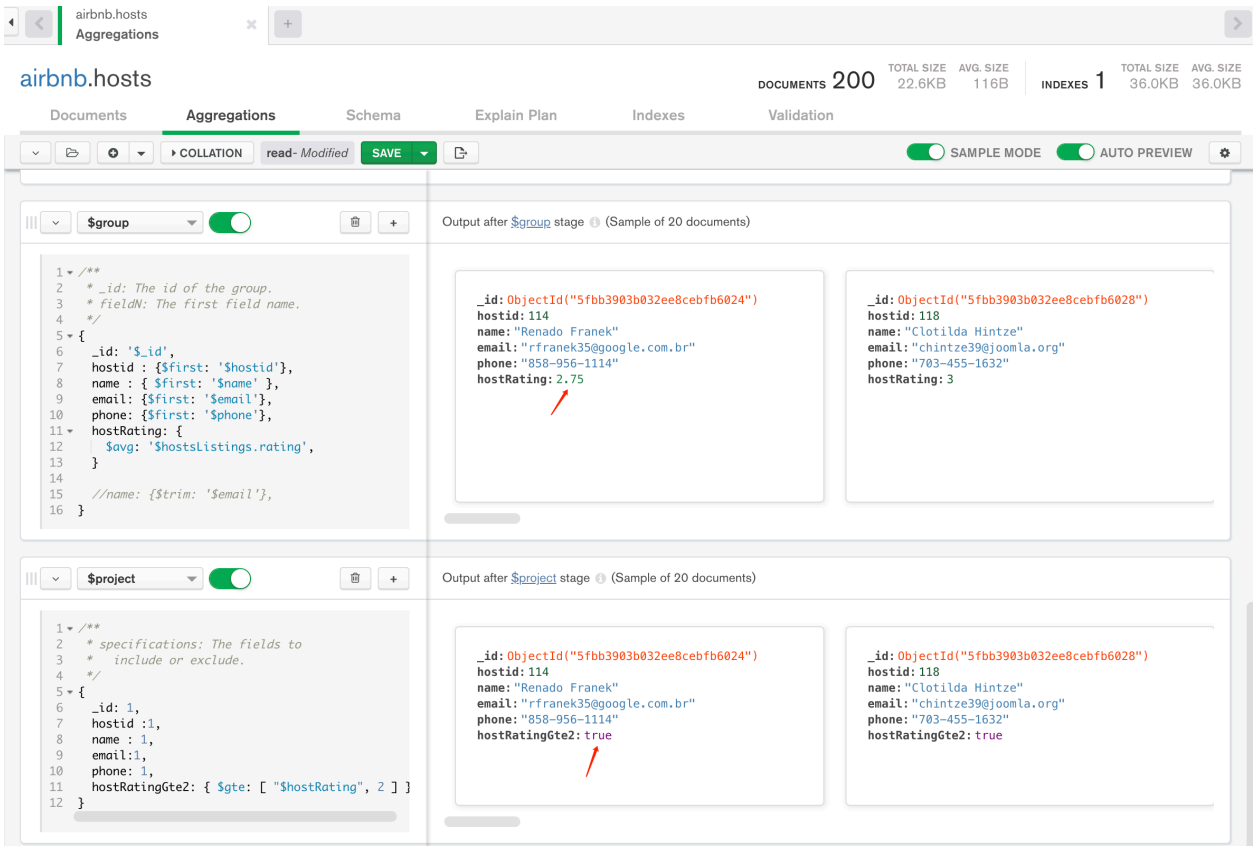


3. One must contain a complex search criterion (more than one expression with logical connectors)

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);
```

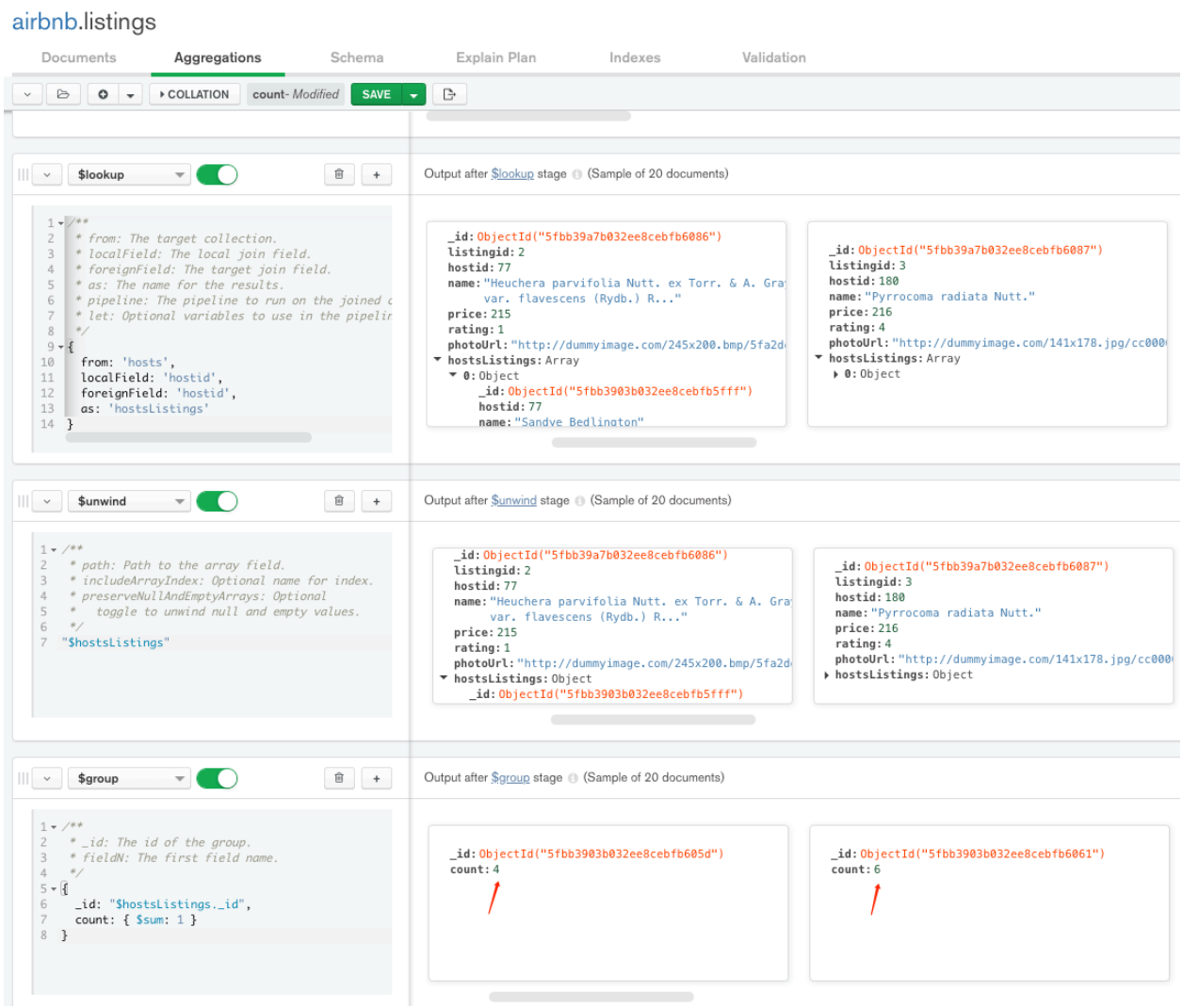


4. One should be counting documents for a specific user

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.listins.aggregate(query);
```

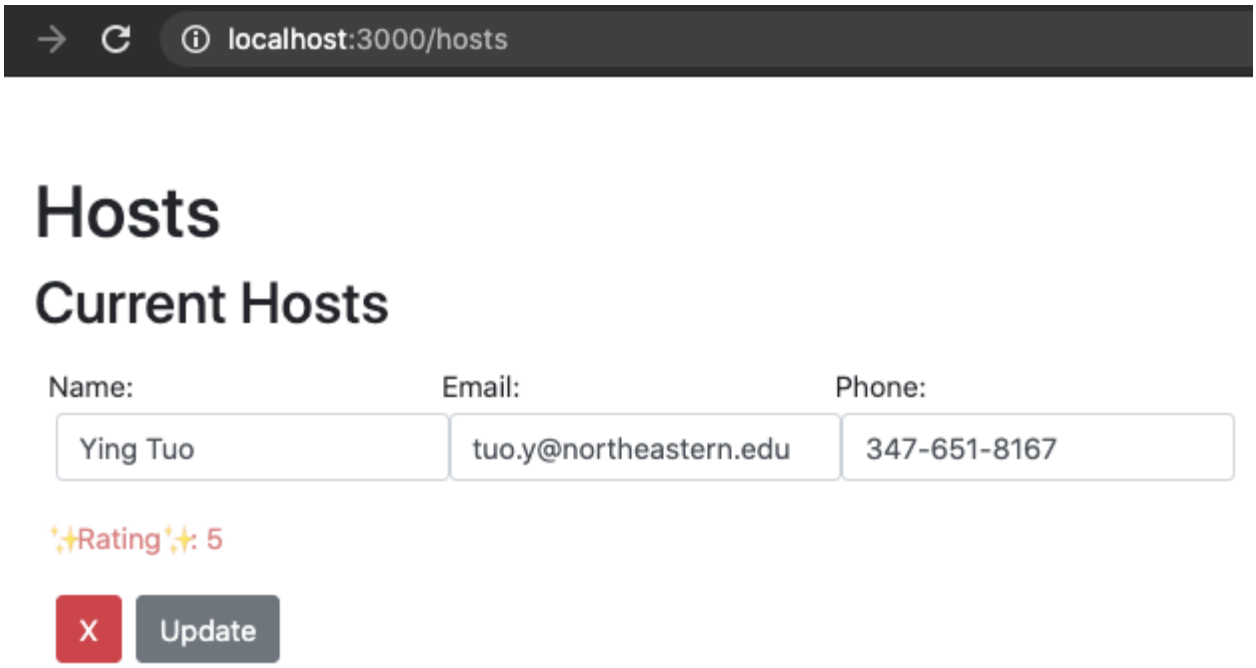


5. One must be updating a document based on a query parameter

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

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

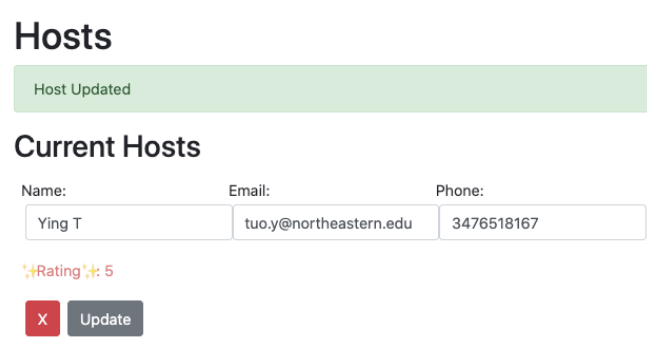
```
      {
        $set: {
          name: host.name,
          email: host.email,
          phone: host.phone
        },
      }
    );
```




```
86  myDB.updateHost = async function (host) {
87    const client = MongoClient(uri, { useUnifiedTopolog
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
```

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE          2: node

POST /hosts/update 302 60.993 ms - 56
GET /hosts 200 498.450 ms - 23196
GET /stylesheets/style.css 304 1.436 ms - -
[nodemon] restarting due to changes...
[nodemon] starting `node ./bin/www`
[nodemon] restarting due to changes...
[nodemon] 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
5fbb45aa64cd1dabe870ebd5
Ying T
tuo.y@northeastern.edu
3476518167
update CommandResult {
  result: { n: 1, nModified: 1, ok: 1 },
```



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