DSA5104 Project—

Database System for Airbnb

Abstract

The objective of this project is to understand the basic process of data management and retrieval in real applications and organizations, which generally covers the following tasks and procedures: data preprocessing and generation, data visualization, data storing in relational and non-relational databases and data querying.

In data preprocessing, we select meaningful features and handle null values. Then we use available tables to generate transaction data. With meaningful datasets, we visualize several relationships to get a deeper understanding of the data. After that, we are using two kinds of databases for storing data and comparing their performances through queries.

Data Introduction

The dataset we choose is Airbnb homestay data in Bangkok, Central Thailand, Thailand. Initially, there are three datasets: calendar, reviews and listing. In specific, the first dataset contains available periods and corresponding prices. Dataset reviews has information about reviewers' name, review's date and comments. As for listing, it contains basic information of the homestay including host name and id, neighborhood name, house position, room type, price and so on. To build a proper and meaningful database, we will filter appropriate attributes and conduct data preprocessing on them.

Data Preprocessing and Generation

(a) Dataset listing

This dataset contains basic information of homestays, and thus, should be paid most attention.

The original data has 17075 rows and 74 columns. At the beginning, we suppose some columns provide little information or have redundancy with other columns and drop these columns. For example, "availability_30" and "availability_60" are dropped because availability of a homestay has been recorded in the dataset calendar. "number_of_reviews" is dropped because this instance can be calculated in table reviews.

After selecting proper columns, it is time to handle null values. We find a row whose host name is "Jakchai" contains several null values. Thus, we log in to the airbnb website and fill these null values to preserve validity of this dataset. Besides, for numerical attributes that contain null values, we replace null values with the mean of that column.

After simple attribute selection and dropping null values, we get a cleaner listing set. Then, with a much deeper understanding of the dataset, we drop columns "neighborhood", "room_type" and "price" because all of them can be acquired or calculated from other tables.

However, we did not drop the properties with abnormally high prices since some hosts may set an abnormal price in the first place.

(b) Dataset calendar and review

After going through tables, calendars and reviews, we find there are no redundant attributes or null values. Thus, we preserve these two datasets and conduct the following data analysis.

(c) Dataset Transaction generation

With the 3 clean datasets, we are lacking a transaction dataset composed with transaction id, customer id, accommodation date and total spend. Thus, we used the review data to generate transaction data.

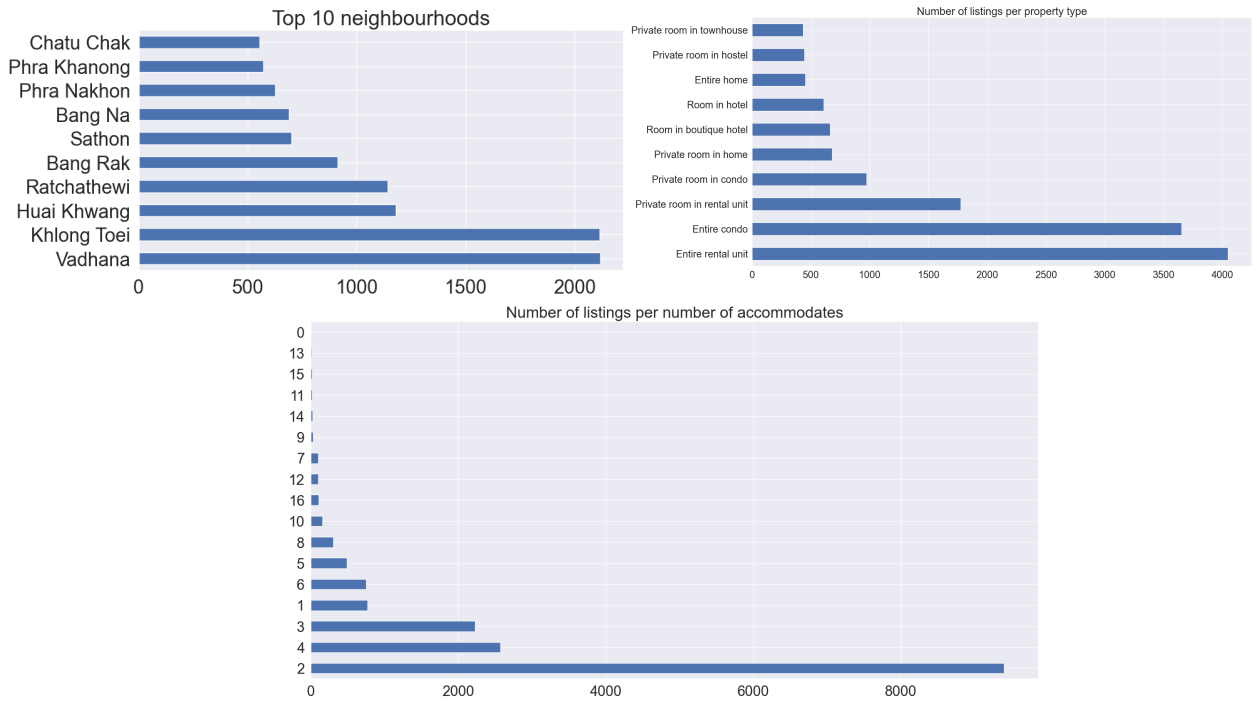
In specific, we use the date from the review table as the end_date. After generating some random number as total_days, we calculated the start_date from end_date. Then we use the adjusted_price from the calendar to generate total_price.

It should be noted that our generated transaction dataset contains no null values. However, in the real world, transaction tables may contain null values.

Data Visualization

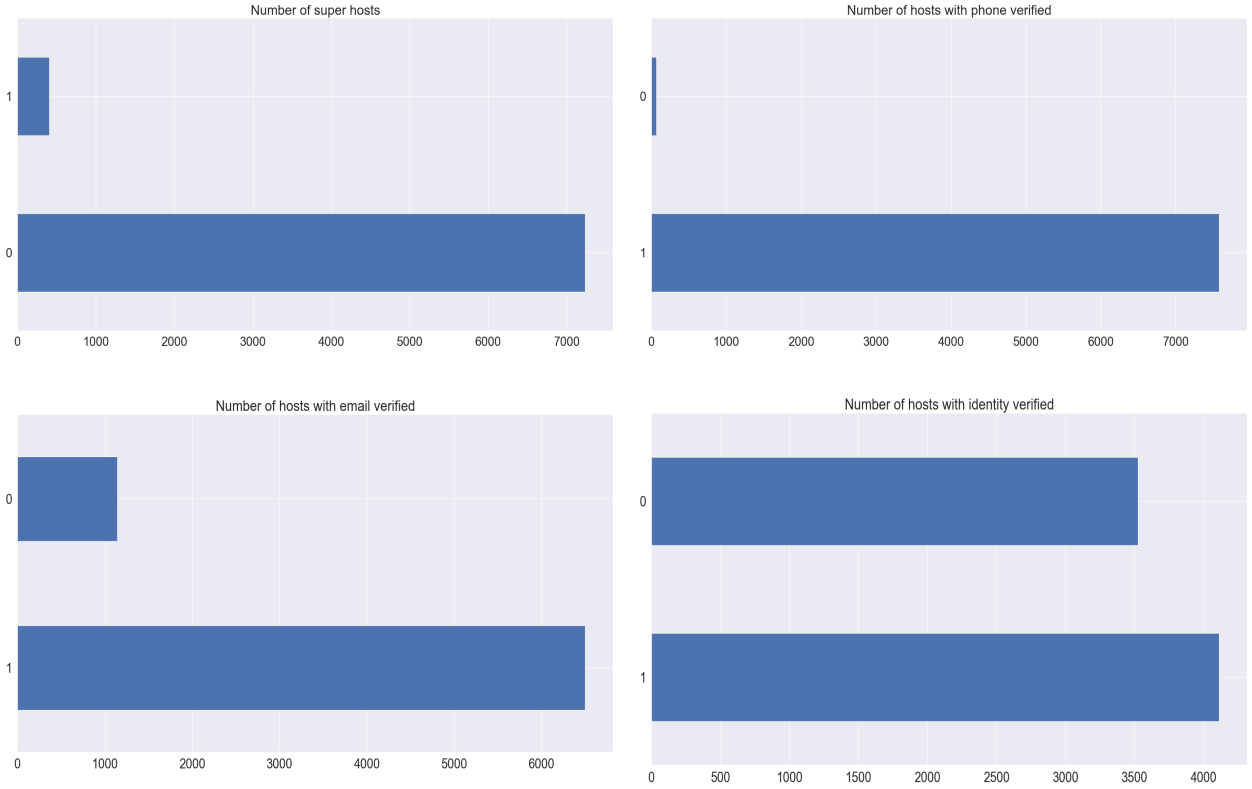
After data generation and preprocessing, here comes data visualization. We would like to gain information about listings, reviews, hosts and price.

(a) Listings information



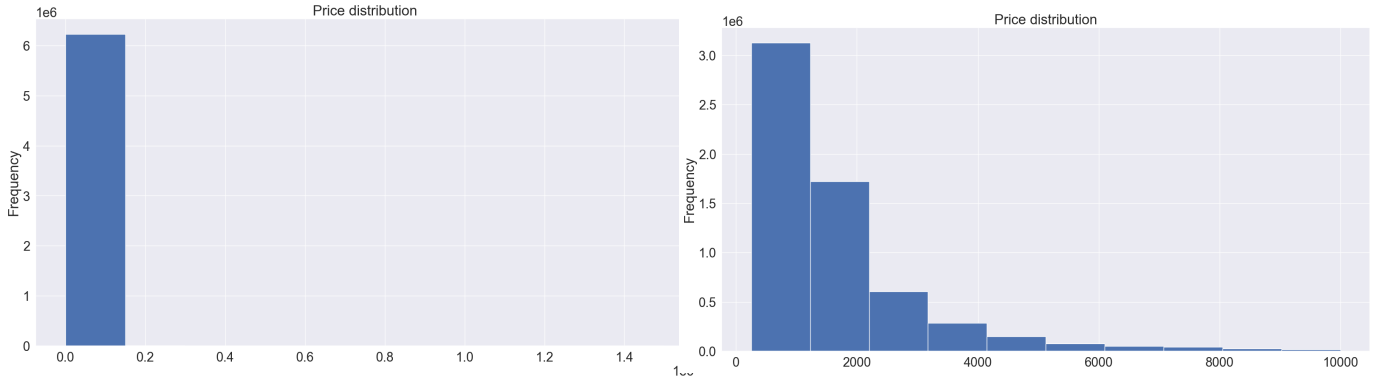
The above three tables show the number of listings per neighborhood, per property type and per number of accommodates. From these graphs, we know that the location with the most neighborhood properties is Vadhana. The two most popular property types are entire condos and entire rental units. Besides, most properties can accommodate two persons and there are also listings that can accommodate 5, 6 or more persons.

(b) Host information



The above four tables are about host information. The first graph shows the number of non-super hosts is heavier than that of super hosts. The second one implies most hosts' phone numbers are verified. Besides, with phone verification, a small proportion of hosts have invalid email. Last, the number of verified and not verified hosts are approximately equal.

(c) House Price information

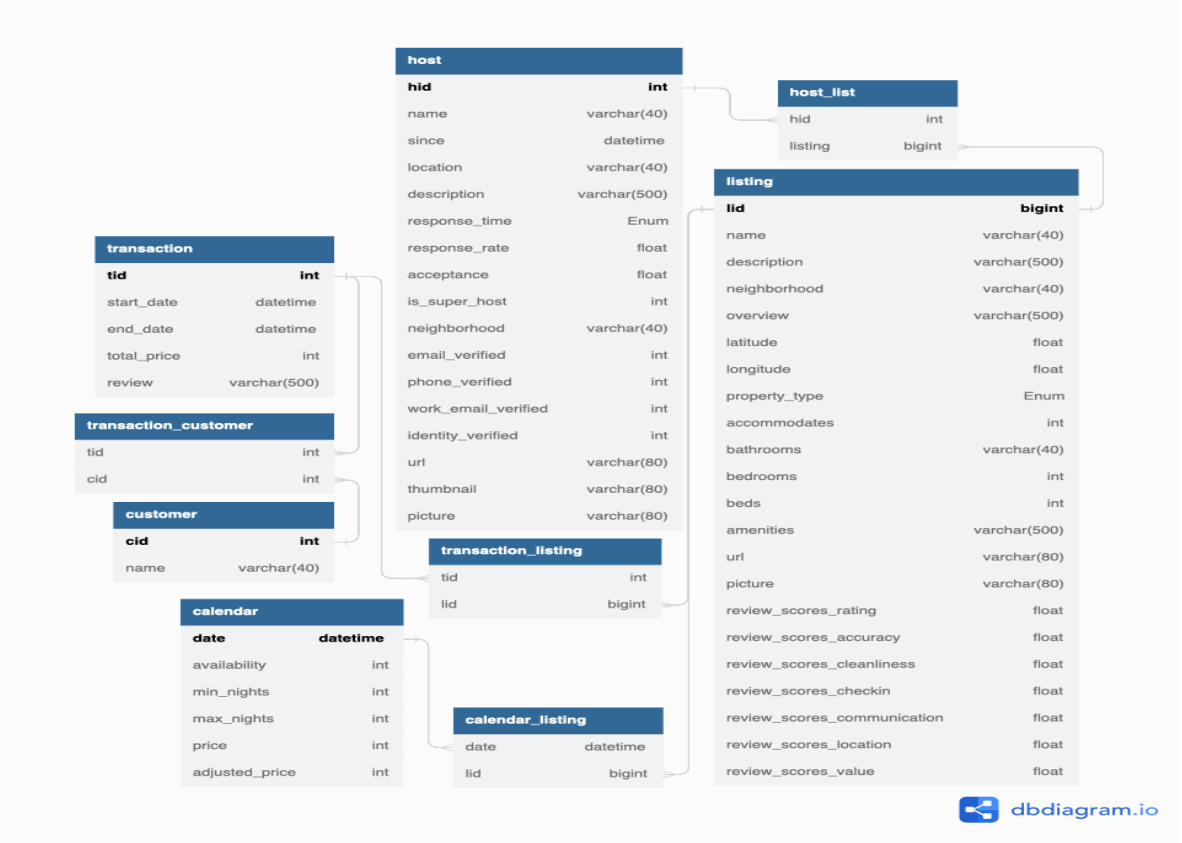


Here comes the price information. When we plot all prices in one graph, the above left one shows concentration in one bar. Such phenomena result from two outlier prices which are extremely large. After we delete the outliers, we get the right graph and conclude that most homestays have prices lower than 2000.

Data Store in Relational Database

Since we generated the new dataset after data cleaning and preprocessing, we are going to design the entity-relationship model. Here, we have a dataset list for ‘calendar’, ‘customer’, ’transaction’, ’host’ and ‘listing’. First of all, there is no doubt that the relation between ‘customer’ and ‘transaction’ is one to many since one customer could have many transactions. Next, we are considering the relationship between ‘host’ and ‘listing’. Normally, one host may post more than one advertisement in his listing. Therefore, it is a ‘one to many’ relation between them. As for relation between ‘transaction’ and ‘listing’, it is a ‘one to one’ relation between them as one transaction is only including one list. Last, the relation between ‘calendar’ and ‘listing’ is ‘one to one’ as well.

Below graph is shown the entity-relationship diagram:



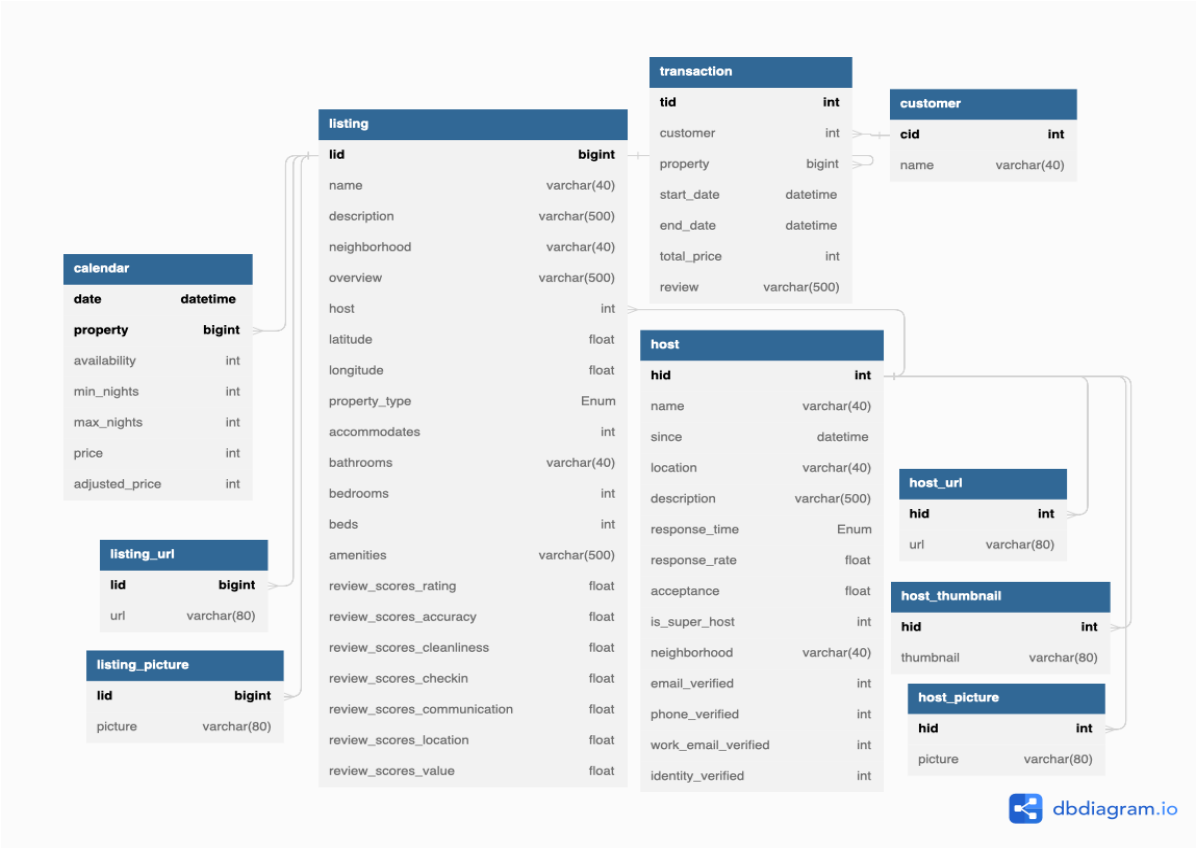
Next step, we are going to design the relation tables based on the rule of BCNF. For the relation table ‘listing’, a functional dependence ‘url’ → ‘ lid’ is a violation, therefore, we have to decompose into the new table called ‘listing_url’, which includes two attribute ‘lid’ and ‘url’. it is the same as another functional dependence ‘picture’ → ‘ lid’, so that we decompose into another new table called ‘listing_picture’, which only includes attribute ‘lid’ and ‘picture’.

For the table ‘host’, there are the same violation issues on functional dependence that ‘url’ → ‘ hid’,‘thumbnail’ → ‘ hid’ and ‘picture’ → ‘ hid’. Therefore, the table ‘host’ can be decomposed into 4 tables, which are ‘host’,’host_url’,’host_thumbnail’ and ‘host_picture’.

We created the total 10 tables: ‘calendar’,’listing’, ‘host_thumbnail’, ’host’, ’listing_picture’, ’host_picture’, ’host_url’, ’transaction’ and ‘customer’ which are all complied with the rule of BCNF.

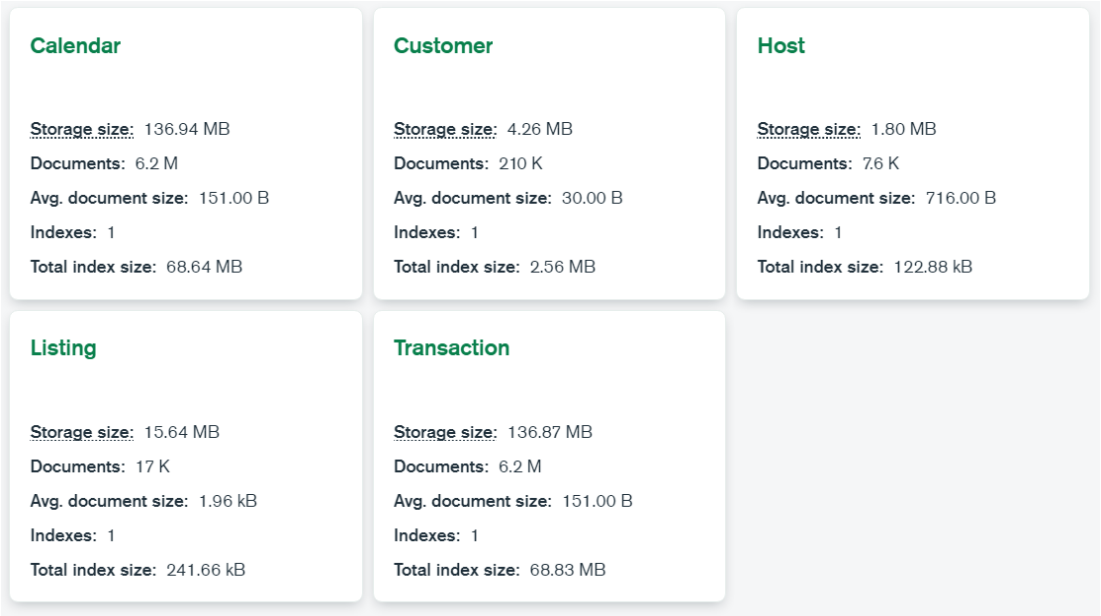
For table ‘calendar’, attribute ‘date’ will be the primary key and foreign key ‘property’ is referenced to attribute ‘ lid’ in table ‘listing’. For table ‘listing_url’ and ‘listing_picture’, both of their primary key ‘lid’ are also referenced to attribute ‘lid’ in table ‘listing’. It is the same as table ‘host’. The primary key ‘hid’ in table ‘host’ is referencing three tables ‘host_url’,’host_thumbnail’ and ‘host_picture’ with the same ‘hid’. After that, the attribute ‘hid’ in table ‘host’ is referenced to table ‘listing’. Based on the ER model relation, there always relation ‘one to one’ between them. Therefore, the attribute ‘lid’ in table ‘listing’ is referenced from attribute ‘property’ in table ‘transaction’ and attribute ‘customer’ is also referenced from attribute ‘cid’ in table ‘customer’.

The relation schema is shown below:



Data Store in Non-relational Database(MongoDB)

Besides storing data in relational database management systems, we also try to store data in non-relational databases to explore a more efficient way. From available options HBase and MongoDB, we choose MongoDB because it is explained in detail in lectures and we get deep understanding in this database system.



We considered various designs for the database. Similar to SQL databases , we decide to use the 5 collections because this schema makes it easy to query for the frequently demanded information.

For Calendars, it's arguable that they could be embedded inside the corresponding Listing document. However, when considering the use case in context, we reckon there are mainly 2 scenarios when such information is queried:

1. The customer wants to see the available dates of a particular listing
2. The customer wants to find all available listings on a particular date

Embedding Calendars inside Listings makes scenarios 1 easy to accomplish. For scenario 2, on the other hand, we need to iterate through all listings and check if they contain the specified date. This is computationally expensive. Thus, we decided to store Calendar in its own collection.

As for Transaction, there are the following 2 scenarios:

1. A customer wants to see his/her/their transaction history
2. A host wants to see all transactions of a particular listing

For the same consideration to facilitate both queries, we decide to store Transaction in its own collection too.

Database Queries

With well-built relational and non-relational databases, we are able to get information efficiently using queries. Through brainstorming requirement analysis, we conclude the following customers' requirements: check property information including availability and comments, search host by name or ratings, give comments and reviews after stay, check use information, etc.

Here is the table listing our requirements, corresponding SQL queries and MongoDB queries.

Natural Language	SQL Query	MongoDB Query
Check property information (Find availability of a certain property on a certain day.)	SELECT availability FROM calendar JOIN listing WHERE listing.name=... AND date=...	listing=db.Listing.findOne({“name”: …}) db.Calendar.find({“property”: listing._id, “date”:...}, {“availibility”:...})
Book available property (Book a property and set its availability to 0 in booking period)	UPDATE calendar SET availability = 0 WHERE property=... AND date>=... AND date<=...	db.Calendar.updateMany({ “property”:listing._id, “date”; {“$gte”: …, “$lte”:...}}, {“$set”:{“availability”:False}})
Search property by name/type/host/…	SELECT * FROM listing WHERE property_type=... AND neighbourhood=...	db.Listing.find({“neighbourhood”: …, “property_type”: …, “bedrooms”: …, “beds”: …})
Give comments and reviews after stay	UPDATE transaction SET review=... WHERE tid=...	db.Transaction.updateOne ({“_id”:...}, {“$set”: {“review”: …}})
Search host by ratings/name… (select a host whose response rate is large and his/her listings has high review scores)	SELECT * FROM host H WHERE response_rate>=... AND EXISTS (SELECT * FROM listing L WHERE L.host=H.hid AND L.review_score>=...)	hosts=db.Listing.find({“review_scores_rating”:{“$gte”: …}}, {“host”:...}) db.Host.find({“_id”: {“$in”: [h.host for h in hosts]}}, “response_rate”: {“$gte:...”})
Check user information	SELECT * FROM customer WHERE name=...	db.Customer. find({“name”: …})

Member Workload

In this project, all of the group members work together. Thus, we collaborate in each part and the workload is distributed evenly.