ADM Final Project

Implementation of Hospitality E-Commerce using MongoDB

Sahitya Reddy Bollavaram 4849759, Sanchayan Bhunia 4849650

The domain that we have chosen is Hospitality E-Commerce. The project is our interpretation of Airbnb's model. We have used the DIBRIS cluster for implementation.

About the Dataset

Dataset – Airbnb Data for Amsterdam

(http://insideairbnb.com/get-the-data.html, https://www.kaggle.com/erikbruin/airbnb-amsterdam)

The dataset consists of

1. listings data
   The listings data consists of

   id,listing_url,scrape_id,last_scraped,name,description,neighborhood_overview,picture_url,host_id,host_url,host_name,host_since,host_location,host_about,host_response_time,host_response_rate,host_acceptance_rate,host_is_superhost,host_thumbnail_url,host_url,host_neighbourhood,host_listings_count,host_total_listings_count,host_verifications,host_has_profile_pic,host_identity_verified,neighbourhood,neighbourhood_cleansed,neighbourhood_group_cleansed,latitude,longitude,property_type,room_type,accommodates,bathrooms,bathrooms_text,bedrooms,beds,amenities,price,minimum_nights,maximum_nights,minimum_minimum_nights,maximum_minimum_nights,minimum_maximum_nights,maximum_maximum_nights,minimum_nights_avg_ntm,maximum_nights_avg_ntm,calendar_updated,has_availability,calendar_last_scraped,number_of_reviews,number_of_reviews_ltm,first_review,last_review,review_scores_rating,review_scores_accuracy,review_scores_cleanliness,review_scores_checkin,review_scores_communication,review_scores_location,review_scores_value,license,instant_bookable,calculated_host_listings_count,calculated_host_listings_count_entire_homes,calculated_host_listings_count_private_rooms,calculated_host_listings_count_shared_rooms,reviews_per_month
2. location information – neighbourhood name, coordinates, country, city
3. amenities information – Amenities present in each property.
4. review details – id, listing id, date, reviewer id, reviewer name, comments,
   review details contain the user id that has reviewed.

Data cleansing and generation:

User data:

The user is an important entity of e-commerce data. Not a lot of information about the users is available in the dataset.

To construct user information, the reviewer id field in review data and host id in listing data is considered to build the user information, along with a few dummy ids (users who have not used the features of the application but are present in the system). To build the missing properties such as last name, gender, country, country code, phone number, email, a library called Faker **https://faker.readthedocs.io/en/master/** is used.

Listings Data:

Since most of the information about the listings is already present in the data set. Unnecessary information (scraping id, date) is removed.

Bookings Data:

Booking Id, Booking date, number of nights, total price(number of nights * price per night) are generated with the help of the data in Listings data.

Review Data:

Although review data contains comments and date, properties such as review_scores_rating,review_scores_accuracy,review_scores_cleanliness,review_scores_checkin,review_scores_communication,review_scores_location,review_scores_value are moved from Listings data to Review data.

A piece of python code (Pandas) is used to build the relations between all the data.

Conceptual Schema:

Entities obtained from the data are as following

1. User Entity
2. Listing Entity
3. Location Entity
4. Amenities
5. Booking Entity
6. Review Entity

Please refer to the ER diagram in the project folder for the complete view.

The relations between the entities are as following.

Entities	Relation
User -> Bookings	One to Many
User -> Listings	One to Many
Listings -> Location	One to One
Listings -> Amenities	One to Many
Listings -> Reviews	One to Many

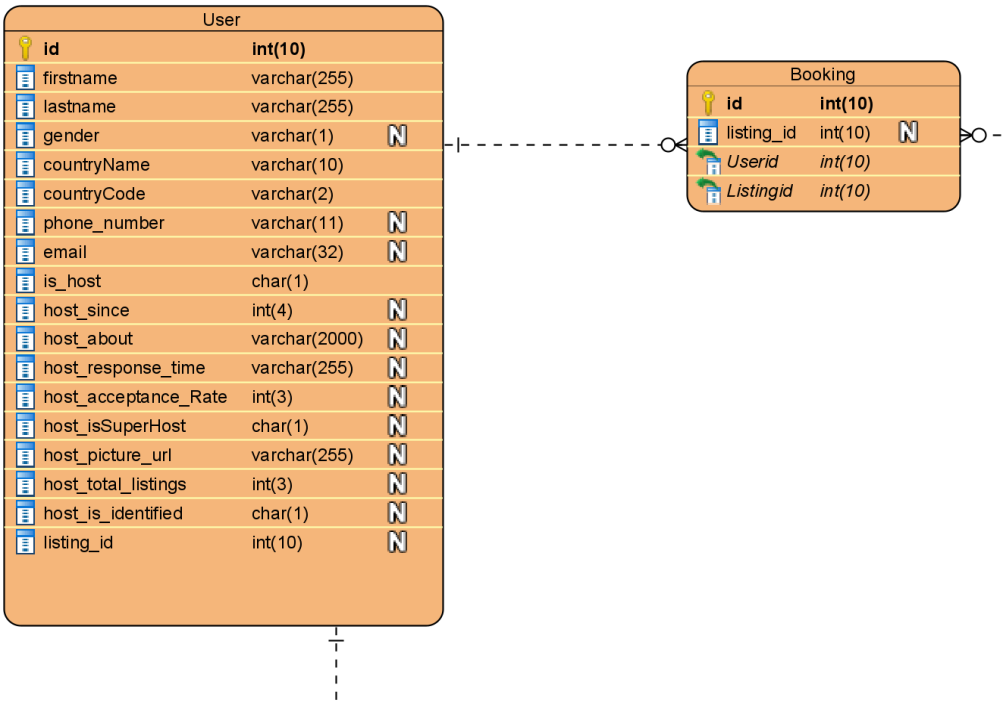
Since redundant data is not a problem, multiple attributes are duplicated in order to achieve Query based design.

User_bookings aggregate

The workload is analyzed as following for creating aggregates.

- Given a User ID, get the user's information.
- (user's first name, last name, email address, phone number, gender, country, if he is a host(is_host), host since, about him, his response time, picture url, identity verified status)
- Given a User ID,get the booking history.(Booking ID, Listing ID(property ID), Total Price, No of Nights, Booking Date)
- User makes a booking.
- Get the number of bookings of each user.

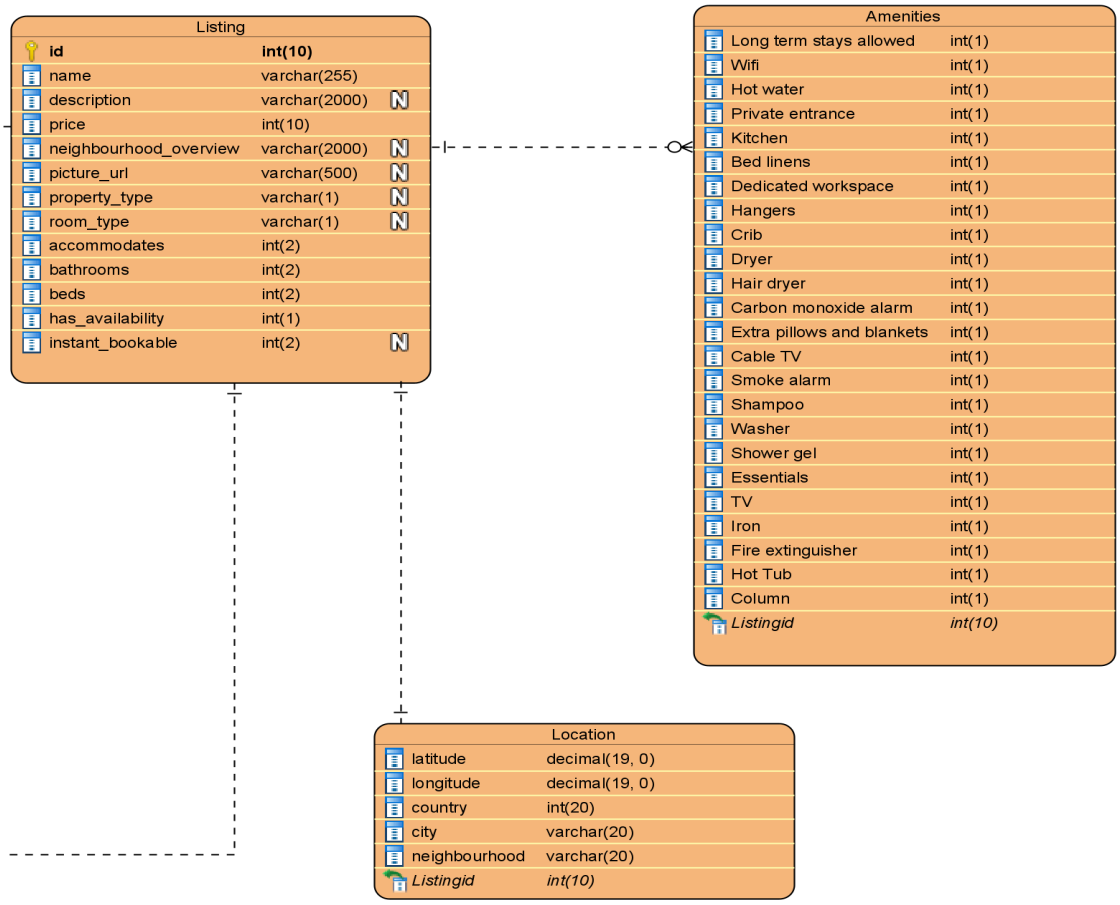
Workload 1,2,3,4 can be achieved from the aggregate below.



Listings aggregate

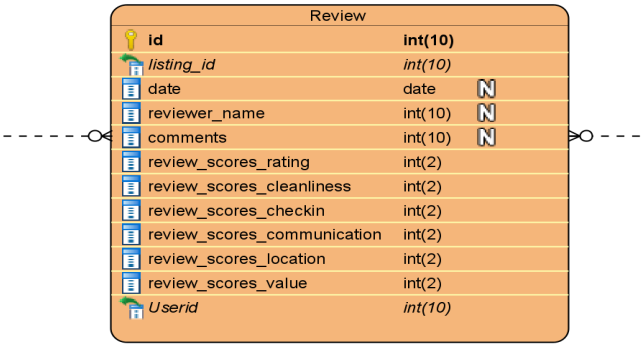
- Given a User ID, get all of listings.
- Get all the Listings' names, prices, picture_urls, based on neighbourhood area and the number of people the property accomodates.
- To the above query, price filter is applied.
- Given User's Coordinates, Get all the listings' name, price, picture_url , coordinates in 1km radius.
- Get the number of properties for each property type.
- Add a new listing and change the user's is_host property to true.

To achieve the above workload on the following aggregate, the associated user_id is stored here.



Reviews aggregate

- User writes a review about one listing (with all review_scores_rating,review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_score_value, review_scores_location).
- Get all reviews of a particular listings ID.
  Storing the associated listing ID would help this workload .
- Get the average rating of all kinds of ratings of each property based (cleanliness, check-in, communication, value, accuracy)
- Order the average rating of neighbourhood area based on user score of the location.
  Storing the neighbourhood area's name would help with this workload.



Storage Model

For addressing the data at physical level, the workload is analysed as following.

1. Given a User ID, get the user's information. (user's first name, last name, email address, phone number, gender, country, if he is a host(is_host), host since, about him, his response time, picture url, identity verified status)

1. User makes a booking.
2. Given a User ID, get the booking history.(Booking ID, Listing ID(property ID), Total Price, No of Nights, Booking Date)
3. Get the number of bookings of each user.

   - For the above workloads, the aggregate is partitioned by _id which the user id. Hash based partition is applied for data distribution, parallel computing and efficient use of RAM.

4. Given a listing Id, get the name, description, location, amenities, number of baths, number of beds, property type, has_availability, instantly bookable.
5. Given a User ID, get all of listings.

6. Get all the Listings' names, prices, picture_urls, based on neighbourhood area and the number of people the property accomodates.
7. To the above query, price filter is applied.
8. Given User's Coordinates, Get all the listings's name, price, picture_url , coordinates in 1km radius.
9. Get the number of properties for each property type.
10. Add a new listing and change the user's is_host property to true.

    - In order to optimize the execution of these workload queries, 2 instances of the data is required. R1, R2. R1 is partitioned by _id which is the listing ID and an index is applied on user ID. This satisfies Workload 1 and 2.
    - R2 is partitioned by neighbourhood's name. A compound index on accommodates and price is applied for 3,4 workloads.
    - For workload 5, a geospatial index is applied on coordinates object. This index can be applied in either R1 or R2.
    - Workload 7 is addressed is explained in the further sections.

11. User writes a review about one listing (with all review_scores_rating,review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_score_value, review_scores_location).
12. Get all reviews of a Listing ID.
13. Get the average rating of all kinds of ratings of each property based (cleanliness, checkin, communication, value, accuracy)
14. Order the average rating of neighbourhood area based on user score of the location.

    - In order to optimize the execution of these workload queries, 2 instances of the data is required. R1, R2.
    - R1 is partitioned by ListingID to satisfy the 2nd workload. R2 is partitioned by neighbourhood area's name for workload 4.

How collections are organized & Why

1) Assessing relationships between entities

The relations below are mentioned using mongo's ways of specifying relations.

Entities	Relation	Explanation	Decision
User -> Bookings	One to Few	Booking information doesn't contain too many nested fields.	To embed bookings is user data as an array of objects. **Embedded Document Pattern**
User -> Listings	One to Many	The application supports read and writes based on various attributes of the listings data. To embed them in user	To store Listings as a separate collection by embedding the user entity's id attribute as user_id.

		data would result is slow performance and indexing array objects is slow.	
Listings -> Location	One to One	-	To embed location in listing's document.
Listings -> Amenities	One to Many	Amenities are stored as an array of string values since they do not have complex data and do not need any indexing.	To embed the Amenities as an array of string values in listing's document.
Listings -> Reviews	One to Squillion	A single Listing can have multiple reviews and its ever growing	To store reviews separately in a new collection by embedding listing's id in each review. **Subset Pattern**

Non-functional Requirements

Below are a few non-functional requirements that are considered for a good user experience (from a data point of view)

1. **The users should be able to always access the application (Availability)**

   MongoDB automatically maintains replica sets, multiple copies of data that are distributed across servers, racks and data centres. Replica sets help prevent database downtime using native replication and automatic failover.
   A replica set consists of multiple replica set members. One member acts as the primary member, and the other members act as secondary members. If the primary member fails for any reason (e.g., hardware failure), one of the secondary members is automatically elected to primary and begins to process all reads and writes.
   Sharding also helps with partial availability. For instance, if one shard is down, the other shards continue to work.

   With the help of

   `sh.status();` (information under shards property brings up the information about replicas)

   ```
   {  "_id" : "rs0",  "host" :
   "rs0/it.unige.dibris.lsccluster.node3:27017,it.unige.dibris.ls
   ccluster.node4:27017,it.unige.dibris.lsccluster.node5:27017",
   "state" : 1 }
   ```

```
        {  "_id" : "rs1",  "host" :
"rs1/it.unige.dibris.lsccluster.node6:27017,it.unige.dibris.ls
ccluster.node7:27017",  "state" : 1 }
        {  "_id" : "rs2",  "host" :
"rs2/it.unige.dibris.lsccluster.node8:27017,it.unige.dibris.ls
ccluster.node9:27017",  "state" : 1 }
```

It is clear from the above information that this cluster has 3 replica sets where rs0 contains a primary and 2 secondary nodes, whereas the others contain a primary and a secondary node each.

2. The application should facilitate high request rate (Scalability)

MongoDB supports horizontal scaling through Sharding, distributing data across several machines and facilitating high throughput operations with large sets of data.

An advantage of horizontal scalability is that it can provide system administrators with the ability to increase capacity on the fly, being only limited by how many machines can be connected successfully. Sharding (Partitioning) allows you to add additional instances to increase capacity when required and also help with load distribution, and concurrent processing

Sharding is achieved in the following way:

Sharding is enabled on database – the name of the database is projectUser2.

```
mongos> sh.enableSharding("projectUser2");
{
        "ok" : 1,
        "operationTime" : Timestamp(1631040629, 3),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1631040629, 3),
                "signature" : {
                        "hash" :
BinData(0,"Z6Iao6SwiQNVVg1AfiXIWA4tdV4="),
                        "keyId" :
NumberLong("6975140761171918849")
                }
        }
```

Please read section “Collections and Sharding” for sharding information on individual collections.

3. Data Consistency

MongoDB is strongly consistent by default . A read immediately after a write always returns the updated value.This is because **MongoDB** is a **single-master** system and all reads go to the primary by default.

By default, the write concern for mongo 4.4 is **w:1** which means that write operation will be acknowledged as successful when the primary in a replica set has successfully executed the write operation.

Based on type of the workload, we have used different read and write concerns (sections below).Based on the table shown below, consistency can be set.

Read Concern	Write Concern	Read own writes	Monotonic reads	Monotonic writes	Writes follow reads
`"majority"`	`"majority"`	✅	✅	✅	✅
`"majority"`	`{ w: 1 }`		✅		✅
`"local"`	`{ w: 1 }`				
`"local"`	`"majority"`			✅	

Collections and Sharding:

Hash Based partitioning is used to promote efficient use of RAM and concurrency.

```
mongos> db.createCollection('users_bookings');
mongos> sh.shardCollection("projectUser2.users_bookings", {"_id" :
"hashed"});

mongos> db.createCollection('listings');
mongos> sh.shardCollection("projectUser2.listings", {
"location.neighbourhood_area": "hashed"});


mongos> db.createCollection("listings_1");
mongos> sh.shardCollection("projectUser2.listings_1", {"_id" :
"hashed"});


mongos> db.createCollection('reviews_1');
mongos> sh.shardCollection("projectUser2.reviews_1", { "listing_id":
"hashed"});

mongos> db.createCollection('reviews');
```

```
mongos> sh.shardCollection("projectUser2.reviews", { "neighbourhood_area":
"hashed"});
```

The output of the data cleansing phase are a 3 json files users_bookings.json, listings.json and review.json. They are copied onto the cluster's ssh with the following commands.

```
scp listings.json user2@130.251.61.97:/home/user2/
scp reviews.json user2@130.251.61.97:/home/user2/
scp users_bookings.json user2@130.251.61.97:/home/user2/
```

Importing data to collections:
Data is importing in the following way from the ssh to the mongo cluster

```
mongoimport --db=projectUser2 --collection=users_bookings --authenticationDatabase
"projectUser2" -u user2 -p 5l7n5z --file=users_bookings.json

mongoimport --db=projectUser2 --collection=listings  --authenticationDatabase
"projectUser2" -u user2 -p 5l7n5z --file=listings.json

mongoimport --db=projectUser2 --collection=reviews_1 --authenticationDatabase
"projectUser2" -u user2 -p 5l7n5z --file=reviews.json
```

Data distribution across hash-based shards.

```
mongos> db.users_bookings.getShardDistribution();

Shard rs1 at
rs1/it.unige.dibris.lsccluster.node6:27017,it.unige.dibris.lsccluster.node7:27017
 data : 44.81MiB docs : 141683 chunks : 2
 estimated data per chunk : 22.4MiB
 estimated docs per chunk : 70841

Shard rs0 at
rs0/it.unige.dibris.lsccluster.node3:27017,it.unige.dibris.lsccluster.node4:27017,it.unige.
dibris.lsccluster.node5:27017
 data : 44.76MiB docs : 141610 chunks : 2
 estimated data per chunk : 22.38MiB
 estimated docs per chunk : 70805

Shard rs2 at
rs2/it.unige.dibris.lsccluster.node8:27017,it.unige.dibris.lsccluster.node9:27017
 data : 44.74MiB docs : 141596 chunks : 2
 estimated data per chunk : 22.37MiB
 estimated docs per chunk : 70798

Totals
 data : 134.31MiB docs : 424889 chunks : 6
 Shard rs1 contains 33.36% data, 33.34% docs in cluster, avg obj size on shard : 331B
 Shard rs0 contains 33.32% data, 33.32% docs in cluster, avg obj size on shard : 331B
Shard rs2 contains 33.3% data, 33.32% docs in cluster, avg obj size on shard : 331B
```

```
mongos> db.listings_1.getShardDistribution();
```

```
Shard rs1 at
rs1/it.unige.dibris.lsccluster.node6:27017,it.unige.dibris.lsccluster.node7:27017
 data : 10.56MiB docs : 5803 chunks : 2
 estimated data per chunk : 5.28MiB
 estimated docs per chunk : 2901

Shard rs0 at
rs0/it.unige.dibris.lsccluster.node3:27017,it.unige.dibris.lsccluster.node4:27017,it.unige.
dibris.lsccluster.node5:27017
 data : 10.87MiB docs : 5917 chunks : 2
 estimated data per chunk : 5.43MiB
 estimated docs per chunk : 2958

Shard rs2 at
rs2/it.unige.dibris.lsccluster.node8:27017,it.unige.dibris.lsccluster.node9:27017
 data : 11.17MiB docs : 6105 chunks : 2
 estimated data per chunk : 5.58MiB
 estimated docs per chunk : 3052

Totals
 data : 32.6MiB docs : 17825 chunks : 6
 Shard rs1 contains 32.39% data, 32.55% docs in cluster, avg obj size on shard : 1KiB
 Shard rs0 contains 33.35% data, 33.19% docs in cluster, avg obj size on shard : 1KiB
 Shard rs2 contains 34.25% data, 34.24% docs in cluster, avg obj size on shard : 1KiB
```

```
mongos> db.listings.getShardDistribution();

Shard rs0 at
rs0/it.unige.dibris.lsccluster.node3:27017,it.unige.dibris.lsccluster.node4:27017,it.unige.
dibris.lsccluster.node5:27017
 data : 9.71MiB docs : 5285 chunks : 3
 estimated data per chunk : 3.23MiB
 estimated docs per chunk : 1761

Shard rs2 at
rs2/it.unige.dibris.lsccluster.node8:27017,it.unige.dibris.lsccluster.node9:27017
 data : 14.36MiB docs : 7881 chunks : 3
 estimated data per chunk : 4.78MiB
 estimated docs per chunk : 2627

Shard rs1 at
rs1/it.unige.dibris.lsccluster.node6:27017,it.unige.dibris.lsccluster.node7:27017
 data : 8.53MiB docs : 4659 chunks : 3
 estimated data per chunk : 2.84MiB
 estimated docs per chunk : 1553


Totals
 data : 32.6MiB docs : 17825 chunks : 9
 Shard rs0 contains 29.78% data, 29.64% docs in cluster, avg obj size on shard : 1KiB
 Shard rs2 contains 44.04% data, 44.21% docs in cluster, avg obj size on shard : 1KiB
 Shard rs1 contains 26.17% data, 26.13% docs in cluster, avg obj size on shard : 1KiB
```

Users would search for listings based on the location, and for reviews for a particular listing id, hence the partition key is applied on neighbourhood area and location.

```
mongos> db.reviews.getShardDistribution();

Shard rs1 at
rs1/it.unige.dibris.lsccluster.node6:27017,it.unige.dibris.lsccluster.node7:27017
 data : 125.23MiB docs : 213619 chunks : 3
 estimated data per chunk : 41.74MiB
 estimated docs per chunk : 71206

Shard rs2 at
rs2/it.unige.dibris.lsccluster.node8:27017,it.unige.dibris.lsccluster.node9:27017
 data : 68.42MiB docs : 118191 chunks : 3
 estimated data per chunk : 22.8MiB
 estimated docs per chunk : 39397

Shard rs0 at
rs0/it.unige.dibris.lsccluster.node3:27017,it.unige.dibris.lsccluster.node4:27017,it.unige.
dibris.lsccluster.node5:27017
 data : 60.84MiB docs : 102632 chunks : 2
 estimated data per chunk : 30.42MiB
 estimated docs per chunk : 51316

Totals
 data : 254.49MiB docs : 434442 chunks : 8
 Shard rs1 contains 49.2% data, 49.17% docs in cluster, avg obj size on shard : 614B
 Shard rs2 contains 26.88% data, 27.2% docs in cluster, avg obj size on shard : 607B
 Shard rs0 contains 23.9% data, 23.62% docs in cluster, avg obj size on shard : 621B
```

```
mongos> db.reviews_1.getShardDistribution();

Shard rs0 at
rs0/it.unige.dibris.lsccluster.node3:27017,it.unige.dibris.lsccluster.node4:27017,it.unige.
dibris.lsccluster.node5:27017
 data : 116.2MiB docs : 198097 chunks : 3
 estimated data per chunk : 38.73MiB
 estimated docs per chunk : 66032

Shard rs2 at
rs2/it.unige.dibris.lsccluster.node8:27017,it.unige.dibris.lsccluster.node9:27017
 data : 90.14MiB docs : 152922 chunks : 3
 estimated data per chunk : 30.04MiB
 estimated docs per chunk : 50974

Shard rs1 at
rs1/it.unige.dibris.lsccluster.node6:27017,it.unige.dibris.lsccluster.node7:27017
 data : 80.19MiB docs : 137176 chunks : 2
 estimated data per chunk : 40.09MiB
 estimated docs per chunk : 68588

Totals
 data : 286.54MiB docs : 488195 chunks : 8
 Shard rs0 contains 40.55% data, 40.57% docs in cluster, avg obj size on shard : 615B
 Shard rs2 contains 31.45% data, 31.32% docs in cluster, avg obj size on shard : 618B
 Shard rs1 contains 27.98% data, 28.09% docs in cluster, avg obj size on shard : 612B
```

Sharding status after sharding the collection and importing the data looks as following

```
sh.status();
```

```
{  "_id" : "projectUser2",  "primary" : "rs0",  "partitioned" : true,  "version" : {
"uuid" : UUID("90fcc92e-5398-4427-8c7c-450a78a8c601"),  "lastMod" : 1 } }
                projectUser2.listings
                        shard key: { "location.neighbourhood_area" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                rs0     3
                                rs1     3
                                rs2     3
                        { "location.neighbourhood_area" : { "$minKey" : 1 } } -->> {
"location.neighbourhood_area" : "Bijlmer-Centrum" } on : rs1 Timestamp(2, 0)
                        { "location.neighbourhood_area" : "Bijlmer-Centrum" } -->> {
"location.neighbourhood_area" : "Centrum-West" } on : rs2 Timestamp(3, 0)
                        { "location.neighbourhood_area" : "Centrum-West" } -->> {
"location.neighbourhood_area" : "De Baarsjes - Oud-West" } on : rs1 Timestamp(4, 0)
                        { "location.neighbourhood_area" : "De Baarsjes - Oud-West" } -->> {
"location.neighbourhood_area" : "De Pijp - Rivierenbuurt" } on : rs2 Timestamp(5, 0)
                        { "location.neighbourhood_area" : "De Pijp - Rivierenbuurt" } -->>
{ "location.neighbourhood_area" : "IJburg - Zeeburgereiland" } on : rs1 Timestamp(6, 0)
                        { "location.neighbourhood_area" : "IJburg - Zeeburgereiland" } -->>
{ "location.neighbourhood_area" : "Oud-Noord" } on : rs2 Timestamp(7, 0)
                        { "location.neighbourhood_area" : "Oud-Noord" } -->> {
"location.neighbourhood_area" : "Watergraafsmeer" } on : rs0 Timestamp(7, 1)
                        { "location.neighbourhood_area" : "Watergraafsmeer" } -->> {
"location.neighbourhood_area" : "Zuid" } on : rs0 Timestamp(5, 2)
                        { "location.neighbourhood_area" : "Zuid" } -->> {
"location.neighbourhood_area" : { "$maxKey" : 1 } } on : rs0 Timestamp(5, 3)
                projectUser2.listings_1
                        shard key: { "_id" : "hashed" }
                        unique: false
                        balancing: true
                        chunks:
                                rs0     2
                                rs1     2
                                rs2     2
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : NumberLong("-
6148914691236517204") } on : rs0 Timestamp(1, 0)
                        { "_id" : NumberLong("-6148914691236517204") } -->> { "_id" :
NumberLong("-3074457345618258602") } on : rs0 Timestamp(1, 1)
                        { "_id" : NumberLong("-3074457345618258602") } -->> { "_id" :
NumberLong(0) } on : rs1 Timestamp(1, 2)
                        { "_id" : NumberLong(0) } -->> { "_id" :
NumberLong("3074457345618258602") } on : rs1 Timestamp(1, 3)
                        { "_id" : NumberLong("3074457345618258602") } -->> { "_id" :
NumberLong("6148914691236517204") } on : rs2 Timestamp(1, 4)
                        { "_id" : NumberLong("6148914691236517204") } -->> { "_id" : {
"$maxKey" : 1 } } on : rs2 Timestamp(1, 5)
                projectUser2.reviews
                        shard key: { "neighbourhood_area" : 1, "listing_id" : 1 }
                        unique: false
                        balancing: true
                        chunks:
                                rs0     2
                                rs1     4
                                rs2     4
                        {
                                "neighbourhood_area" : { "$minKey" : 1 },
                                "listing_id" : { "$minKey" : 1 }
                        } -->> { "neighbourhood_area" : "Bijlmer-Centrum", "listing_id" :
954633 } on : rs2 Timestamp(2, 0)
```

```
                        { "neighbourhood_area" : "Bijlmer-Centrum", "listing_id" : 954633 }
-->> { "neighbourhood_area" : "Centrum-Oost", "listing_id" : 43202758 } on : rs2
Timestamp(5, 6)
                        { "neighbourhood_area" : "Centrum-Oost", "listing_id" : 43202758 }
-->> { "neighbourhood_area" : "Centrum-West", "listing_id" : 23416497 } on : rs2
Timestamp(5, 7)
                        { "neighbourhood_area" : "Centrum-West", "listing_id" : 23416497 }
-->> { "neighbourhood_area" : "De Baarsjes - Oud-West", "listing_id" : 1182522 } on : rs2
Timestamp(5, 8)
                        { "neighbourhood_area" : "De Baarsjes - Oud-West", "listing_id" :
1182522 } -->> { "neighbourhood_area" : "De Pijp - Rivierenbuurt", "listing_id" : 5087114 }
on : rs1 Timestamp(5, 4)
                        { "neighbourhood_area" : "De Pijp - Rivierenbuurt", "listing_id" :
5087114 } -->> { "neighbourhood_area" : "De Pijp - Rivierenbuurt", "listing_id" : 31393518
} on : rs1 Timestamp(5, 5)
                        { "neighbourhood_area" : "De Pijp - Rivierenbuurt", "listing_id" :
31393518 } -->> {
                                "neighbourhood_area" : "Oostelijk Havengebied - Indische
Buurt",
                                "listing_id" : 7645716
                        } on : rs1 Timestamp(5, 3)
                        {
                                "neighbourhood_area" : "Oostelijk Havengebied - Indische
Buurt",
                                "listing_id" : 7645716
                        } -->> { "neighbourhood_area" : "Oud-Oost", "listing_id" : 5000874
} on : rs1 Timestamp(4, 3)
                        { "neighbourhood_area" : "Oud-Oost", "listing_id" : 5000874 } -->>
{ "neighbourhood_area" : "Zuid", "listing_id" : 3566834 } on : rs0 Timestamp(4, 1)
                        { "neighbourhood_area" : "Zuid", "listing_id" : 3566834 } -->> {
                                "neighbourhood_area" : { "$maxKey" : 1 },
                                "listing_id" : { "$maxKey" : 1 }
                        } on : rs0 Timestamp(1, 3)
                projectUser2.users_bookings
                        shard key: { "_id" : "hashed" }
                        unique: false
                        balancing: true
                        chunks:
                                rs0     2
                                rs1     2
                                rs2     2
                        { "_id" : { "$minKey" : 1 } } -->> { "_id" : NumberLong("-
6148914691236517204") } on : rs0 Timestamp(1, 0)
                        { "_id" : NumberLong("-6148914691236517204") } -->> { "_id" :
NumberLong("-3074457345618258602") } on : rs0 Timestamp(1, 1)
                        { "_id" : NumberLong("-3074457345618258602") } -->> { "_id" :
NumberLong(0) } on : rs1 Timestamp(1, 2)
                        { "_id" : NumberLong(0) } -->> { "_id" :
NumberLong("3074457345618258602") } on : rs1 Timestamp(1, 3)
                        { "_id" : NumberLong("3074457345618258602") } -->> { "_id" :
NumberLong("6148914691236517204") } on : rs2 Timestamp(1, 4)
                        { "_id" : NumberLong("6148914691236517204") } -->> { "_id" : {
"$maxKey" : 1 } } on : rs2 Timestamp(1, 5)
```

Workload Implementation

From the workload queries defined above, the application is read/write intensive.

1. **Given a User ID, get the user's information.**

(user's first name, last name, email address, phone number, gender, country, if he is a host(is_host), host since, about him, his response time, picture url, identity verified status)

Since MongoDB allows "similar" structures, and attributes that are not present can be left undefined, this approach has been used to show users who are hosts and users who are not hosts.

For mongo version 4.4, the default read concern value is "**available**". However, using "**local**" **readConcern** is a better idea on sharded collections because with sharding, most reads rely on the shard versioning protocol for up to date routing(in case of routing info mismatch, the config servers are contacted for latest routing information).But with "available" read concern this protocol is by passed and it causes inconsistencies in data.

```
mongos>  db.users_bookings.find({"_id": 37404},{first_name:1,last_na
me:1,gender:1,country:1,country_code:1,phone_number:1,
        phone_number:1,email:1,is_host:1, host_about:1, host_respons
e_time:1, host_response_rate:1, host_is_superhost:1, host_picture_ur
l:1,host_identity_verified:1 }).readConcern("local").pretty();
{
        "_id" : 37404,
        "first_name" : "Thomas R.",
        "last_name" : "Fahey",
        "gender" : "F",
        "country" : "United States",
        "country_code" : "US",
        "phone_number" : "+1 45875127419",
        "email" : "ma0fjo7gtz851jkxyoui@yahoo.com",
        "is_host" : false
}

mongos>db.users_bookings.find({"_id": 30390},{first_name:1,last_name:1,gen
der:1,country:1,country_code:1,phone_number:1,
        phone_number:1,email:1,is_host:1, host_about:1, host_respons
e_time:1, host_response_rate:1, host_is_superhost:1, host_picture_ur
l:1,host_identity_verified:1 }).readConcern("local").pretty();

{
        "_id" : 30390,
        "first_name" : "Federica",
        "last_name" : "Herman",
        "gender" : "F",
```

```
        "country" : "Netherlands",
        "country_code" : "NL",
        "phone_number" : "+31 87429999835",
        "email" : "klf3d7k7y7wfc85i7zga@gmail.com",
        "is_host" : true,
        "host_about" : "Hello!\r\nI'm an italian architect living in
The Netherlands!",
        "host_response_time" : "a few days or more",
        "host_response_rate" : "0%",
        "host_is_superhost" : 0,
        "host_picture_url" :
"https://a0.muscache.com/im/pictures/user/eabcd253-a46d-4b21-b735-
76b67d94afdb.jpg?aki_policy=profile_x_medium",
        "host_identity_verified" : 1
}
```


User ID 37404 is not a a host, `30390` is a host and its respective document has additional attributes. The above queries are done on _id and since the partition ID is also _id, the query runs efficiently.

2. **User makes a booking**

Bookings property inside users_bookings is an array and hence **$push** is used to push the new document. Since bookings are important, w: majority is used for majority acknowledgment from members.

```
db.users_bookings.updateOne(
   {"_id": 37404},
   {'$push' : {bookings: {"_id": 340983 , listing_id: 340992, total_price:140, number_of_nig
hts:2, booking_date: "08/09/2021"} }
   },{writeConcern: {w:'majority'}});

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

3. **Given a User ID, get the booking history.(each booking has Booking ID, Listing ID(property ID), Total Price, No of Nights, Booking Date)**

```
db.users_bookings.findOne({"_id": 37404}, {bookings: 1});
{
    "_id" : 37404,
    "bookings" : [
        {
            "_id" : 389565,
            "listing_id" : 28976111,
            "total_price" : 150,
            "no_of_nights" : 2,
            "booking_date" : "28/04/2019"
```

```
        },
        {
            "_id" : 340983,
            "listing_id" : 340992,
            "total_price" : 140,
            "number_of_nights" : 2,
            "booking_date" : "08/09/2021"
        }
    ]
}
```

4. **Get the number of bookings of each listing in descending order. ($unwind)**

```
db.users_bookings.aggregate( [
  {"$unwind": "$bookings"},
  { "$group": { "_id": "$bookings.listing_id", "noOfBookings": { '$sum': 1 }, } },
  { "$sort" : {noOfBookings: -1 } }
  ] );
```

```
{ "_id" : 82482, "noOfBookings" : 860 }
{ "_id" : 802052, "noOfBookings" : 798 }
{ "_id" : 785432, "noOfBookings" : 783 }
{ "_id" : 1814121, "noOfBookings" : 772 }
{ "_id" : 35632344, "noOfBookings" : 707 }
{ "_id" : 694850, "noOfBookings" : 702 }
{ "_id" : 35927687, "noOfBookings" : 695 }
{ "_id" : 851044, "noOfBookings" : 691 }
{ "_id" : 1247334, "noOfBookings" : 686 }
{ "_id" : 654931, "noOfBookings" : 682 }
{ "_id" : 68290, "noOfBookings" : 659 }
{ "_id" : 4210531, "noOfBookings" : 620 }
{ "_id" : 608432, "noOfBookings" : 619 }
{ "_id" : 4449764, "noOfBookings" : 617 }
{ "_id" : 6441316, "noOfBookings" : 608 }
{ "_id" : 193038, "noOfBookings" : 598 }
{ "_id" : 7276869, "noOfBookings" : 595 }
{ "_id" : 68873, "noOfBookings" : 579 }
{ "_id" : 655216, "noOfBookings" : 578 }
{ "_id" : 2497346, "noOfBookings" : 577 }
```

5. **Given a Listing id, get the name, description, location, amenities, number of baths, number of beds, property type, has_availability, instantly bookable.**

```
db.listings_1.findOne({
  _id : 340992
```

```
    },
    {name :1 , decription : 1, location : 1, amenities : 1, bathrooms_text : 1, beds : 1, bedro
    oms : 1, property_type : 1, has_availability : 1, instantly_bookable: 1});
{
    "_id" : 340992,
    "location" : {
        "country" : "NL",
        "city" : "Amsterdam",
        "neighbourhood_area" : "Centrum-Oost",
        "coords" : [
            52.37,
            3.89393
        ]
    },
    "name" : "Stylish  light apartment in de pijp",
    "has_availability" : true,
    "instantly_bookable" : false,
    "property_type" : "Entire apartment",
    "bathrooms_text" : "1 bath",
    "bedrooms" : 1,
    "beds" : 1,
    "amenities" : [
        "wifi",
        "kitchen"
    ]
}
```

6. **Get listings of a user id.**

An **index** is created on **user_id** on the collection listings_1. For this query either of the collections(listings and listings_1 can be used).

```
db.listings_1.createIndex( { user_id: 1 } );
mongos> db.listings_1.createIndex( { user_id: 1 } );
{
        "raw" : {

"rs2/it.unige.dibris.lsccluster.node8:27017,it.unige.dibris.lsccluster.node9:27017
" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 2,
                        "numIndexesAfter" : 3,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
    },

"rs1/it.unige.dibris.lsccluster.node6:27017,it.unige.dibris.lsccluster.node7:27017
" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 2,
                        "numIndexesAfter" : 3,
```

```
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                },

"rs0/it.unige.dibris.lsccluster.node3:27017,it.unige.dibris.lsccluster.node4:27017
,it.unige.dibris.lsccluster.node5:27017" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 2,
                        "numIndexesAfter" : 3,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1631275462, 7),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1631275462, 7),
                "signature" : {
                        "hash" : BinData(0,"R+w8ObhYlRljlEMfvFGbW2UJ8Ms="),
                        "keyId" : NumberLong("6975140761171918849")
                }
```

```
db.listings_1.find({"user_id":  11812316});
```

Execution stats without Index	Execution stats with Index

"executionStats" : {
"nReturned" : 2,
"executionTimeMillis" : 8,
"totalKeysExamined" : 0,
"totalDocsExamined" : 17827,
"shards" : [
{
"shardName" : "rs0",
"executionSuccess" : true,
"nReturned" : 0,
"executionTimeMillis" : 6,
"totalKeysExamined" : 0,
"totalDocsExamined" : 5917,
}
{
"shardName" : "rs1",
"executionSuccess" : true,
"nReturned" : 1,
"executionTimeMillis" : 6,
"totalKeysExamined" : 0,
"totalDocsExamined" : 5804
}
{
"shardName" : "rs2",
"executionSuccess" : true,
"nReturned" : 1,
"executionTimeMillis" : 7,
"totalKeysExamined" : 0,
"totalDocsExamined" : 6106
}]
	"executionStats" : {
	"nReturned" : 2,
	"executionTimeMillis" : 2,
	"totalKeysExamined" : 2,
	"totalDocsExamined" : 2,
	"shards" : [
	{
	"shardName" : "rs0",
	"executionSuccess" : true,
	"nReturned" : 0,
	"executionTimeMillis" : 2,
	"totalKeysExamined" : 0,
	"totalDocsExamined" : 0,
	}
	{
	"shardName" : "rs1",
	"executionSuccess" : true,
	"nReturned" : 1,
	"executionTimeMillis" : 2,
	"totalKeysExamined" : 1,
	"totalDocsExamined" : 1,
	}
	{
	"shardName" : "rs2",
	"executionSuccess" : true,
	"nReturned" : 1,
	"executionTimeMillis" : 2,
	"totalKeysExamined" : 1,
	"totalDocsExamined" : 1,
	}]

The totalDocsExamined across shards is 17827 which is the same as size the entire collection.

```
mongos> db.listings_1.count();
17827
```

On creating an index on user_id, the number of documents examined has dropped to 2 from 17827 and the execution time has dropped to 2ms. The number of shards and execution time for each shard can be seen in the above table.

7. **Get all the Listings' names, prices, picture_urls, based on neighbourhood area and the number of people the property accomodates. (if the requested page number 1, return 10 first records, else return next 10) (using $limit and $skip)**

Properties that are available based on Location and no. of people that would live in the property are returned. Since most search features on ecommerce websites involve pagination, only a defined number of results are shown on the UI. In this case 10 results. Depending on the page number requested by the user, the results are returned.
A **compound index** on accommodates and price is done as following for optimization.

```
mongos> db.listings.createIndex( { 'accommodates': 1 ,'price' : 1} );
{
        "raw" : {

"rs2/it.unige.dibris.lsccluster.node8:27017,it.unige.dibris.lsccluster.nod
e9:27017" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 2,
                        "numIndexesAfter" : 3,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                },

"rs1/it.unige.dibris.lsccluster.node6:27017,it.unige.dibris.lsccluster.nod
e7:27017" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 2,
                        "numIndexesAfter" : 3,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                },

"rs0/it.unige.dibris.lsccluster.node3:27017,it.unige.dibris.lsccluster.nod
e4:27017,it.unige.dibris.lsccluster.node5:27017" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 2,
                        "numIndexesAfter" : 3,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
```

```
        "operationTime" : Timestamp(1631475125, 5),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1631475125, 5),
                "signature" : {
                        "hash" :
BinData(0,"QmgAFBJ/DYSuxXvIsGJCP2MAdnk="),
                        "keyId" : NumberLong("6975140761171918849")
                }
        }
}
```

For page 1:

```
db.listings.find(
{ 'location.neighbourhood_area' : "Zuid", accommodates: 2},
{name : 1, picture_url: 1, price : 1},
{'$limit': 10});
```

For page 2:
The first 10 records are skipped and the next 10 records are retrieved. For page 3, the first 20 records will be skipped and so on.

```
db. listings.find({ 'location.neighbourhood_area' : "Zuid", accommodates: 2},{name : 1, pi
cture_url: 1, price : 1}).skip(10).limit(10);
```


```
{ "_id" : 5299925, "name" : "Apartment a stone's throw from the Rijksmuseam!",
"picture_url" : "https://a0.muscache.com/pictures/ae90e473-6674-42fd-ba11-
a437259faf08.jpg", "price" : 99 }
{ "_id" : 8963247, "name" : "Home 200 mtr. from Museumsquare", "picture_url" :
"https://a0.muscache.com/pictures/ce33a349-20b2-4123-8dfb-cce0593f5abf.jpg", "price"
: 75 }
{ "_id" : 1018703, "name" : "Cosy private studio on houseboat in the centre!*",
"picture_url" : "https://a0.muscache.com/pictures/73004ad9-e8be-4e5c-a8f1-
d947e4637dc0.jpg", "price" : 86 }
{ "_id" : 1102657, "name" : "Studio with balcony near Vondelpark", "picture_url" :
"https://a0.muscache.com/pictures/3408018e-70fc-4ed9-b327-7a5ca1297abe.jpg",
"price" : 60 }
{ "_id" : 9816257, "name" : "Houseboat studio with canal view and bikes", "picture_url" :
"https://a0.muscache.com/pictures/b24d9b8a-6173-47ae-9269-ad4e8faf3afe.jpg",
"price" : 77 }
{ "_id" : 16378597, "name" : "Bright and cosy apartment", "picture_url" :
"https://a0.muscache.com/pictures/13425234-bdf4-4ddb-92b5-17fd9b148dc5.jpg",
"price" : 75 }
{ "_id" : 20127193, "name" : "Stylish boho apartment in Amsterdam South", "picture_url"
: "https://a0.muscache.com/pictures/1aa952aa-0107-47e1-a275-100735561e32.jpg",
"price" : 95 }
{ "_id" : 85008, "name" : "Museum district/Vondelpark studio-apartment", "picture_url" :
"https://a0.muscache.com/pictures/miso/Hosting-85008/original/41e6e043-f6dc-41cb-
b3cf-134c33527aa5.jpeg", "price" : 66 }
```

{ "_id" : 43980, "name" : "View into park / museum district (long/short stay)", "picture_url" : "https://a0.muscache.com/pictures/923eae51-40df-4783-b2b1-7032966a23a3.jpg", "price" : 65 }
{ "_id" : 97476, "name" : "Urban Oasis~beautiful street in Old South", "picture_url" : "https://a0.muscache.com/pictures/823bec74-2169-40f3-a1cc-485c92d3c847.jpg", "price" : 99 }
{ "_id" : 213721, "name" : "Amsterdam Apartment (waterview)", "picture_url" : "https://a0.muscache.com/pictures/85547760/4de4bff9_original.jpg", "price" : 151 }
{ "_id" : 272597, "name" : "Luxury appartm, good neighbourhood", "picture_url" : "https://a0.muscache.com/pictures/2745454/242ed6d2_original.jpg", "price" : 145 }
{ "_id" : 1630603, "name" : "Charming quite room with fast Wifi", "picture_url" : "https://a0.muscache.com/pictures/33a2e342-018e-42c9-be4a-6105f742d648.jpg", "price" : 41 }
{ "_id" : 1057494, "name" : "Lovely bright room with balcony and beatiful view!", "picture_url" : "https://a0.muscache.com/pictures/16397337/5539c9c0_original.jpg", "price" : 79 }
{ "_id" : 447151, "name" : "creative ground floor house close to Vondelpark", "picture_url" : "https://a0.muscache.com/pictures/054dc658-909e-48c3-8cd1-74b358cdf7d8.jpg", "price" : 105 }
{ "_id" : 32325478, "name" : "Spacious room in a bright and big apartment", "picture_url" : "https://a0.muscache.com/pictures/7e0e8e6f-69ea-4e50-aa3e-bad455d572cd.jpg", "price" : 60 }
{ "_id" : 11962960, "name" : "â˜†stylish and spacious Apt in great area w balconyâ˜†", "picture_url" : "https://a0.muscache.com/pictures/miso/Hosting-11962960/original/efa831f1-dabd-4a12-b780-810bff53eb05.jpeg", "price" : 103 }
{ "_id" : 12868928, "name" : "cosy apartment Amsterdam south", "picture_url" : "https://a0.muscache.com/pictures/bd01b991-384f-4b00-89a2-71ccd728fff0.jpg", "price" : 100 }
{ "_id" : 28375507, "name" : "Amsterdam Penthouse Apartment", "picture_url" : "https://a0.muscache.com/pictures/3a25ff00-2df0-4082-97fa-5bb1a0db628d.jpg", "price" : 150 }
{ "_id" : 14752749, "name" : "Spacious an bright apartment next to Vondelpark", "picture_url" : "https://a0.muscache.com/pictures/3711676c-98b0-47a8-80b9-6f3b10606945.jpg", "price" : 120 }

8. **To workload 7 , price filter is applied. ($lt, $gt)**

```
mongos>  db.listings.find({
        'location.neighbourhood_area': 'Bijlmer-Centrum', 'accommodates' :2,
         'price': {
        $lt: 100,
        $gt : 0
        }
        },
        {
        name : 1, picture_url: 1, price : 1
        }
        ).limit(10);
```

```
{
        "_id" : 954629,
        "name" : "Stylish, clean studio, high service",
        "picture_url" : "https://a0.muscache.com/pictures/bad11e40-3e1b-4410-af9a-
1623fe6cfb2e.jpg",
        "price" : 72
}
{
        "_id" : 954633,
        "name" : "Private studio 25m2 for two near A'dam Arena",
        "picture_url" : "https://a0.muscache.com/pictures/6f98344b-5c41-4cac-8f9e-
2462c346fe07.jpg",
        "price" : 79
}
{
        "_id" : 5875932,
        "name" : "Nice place to saty, to enjoy! :)",
        "picture_url" : "https://a0.muscache.com/pictures/270187b1-6ba6-4cc1-9f76-
53995c51d2b6.jpg",
        "price" : 65
}
{
        "_id" : 4336334,
        "name" : "Cozy room for two",
        "picture_url" : "https://a0.muscache.com/pictures/9a014ee9-6741-4039-8d12-
876eef1b31f1.jpg",
        "price" : 55
}
{
        "_id" : 16822162,
        "name" : "Very big private room > center/airport  nearby!",
        "picture_url" : "https://a0.muscache.com/pictures/f838c467-f437-4ad4-905a-
b085ced472a2.jpg",
        "price" : 50
}
{
        "_id" : 27971206,
        "name" : "Modern and cozy room 20 mins by metro from centre",
        "picture_url" : "https://a0.muscache.com/pictures/87bb4034-b85d-4945-8a48-
6ed540f8c5fd.jpg",
        "price" : 49
}
{
        "_id" : 19495563,
        "name" : "GREAT APARTMENT CLOSE TO THE CENTRE IN AMSTERDAM",
        "picture_url" : "https://a0.muscache.com/pictures/75f49081-3d21-4d94-95e1-
78437e201582.jpg",
        "price" : 45
}
{
        "_id" : 43976021,
        "name" : "Bright and spacious bedroom close to metro",
        "picture_url" : "https://a0.muscache.com/pictures/miso/Hosting-
43976021/original/404c87b8-ea99-47b9-ba17-3971042e3a97.jpeg",
        "price" : 35
}
{
        "_id" : 15105977,
        "name" : "Lovely private 35m2 studio in Amsterdam",
        "picture_url" : "https://a0.muscache.com/pictures/7e99d902-b2cf-43c3-9f55-
9d1b4666992f.jpg",
        "price" : 87
}
{
        "_id" : 24528823,
        "name" : "Cosy apartment in Amsterdam near metro",
```

```
        "picture_url" : "https://a0.muscache.com/pictures/10e5b96a-4816-45a2-9d2c-
97d96290da43.jpg",
        "price" : 88
}
```

9. **Given User's Coordinates, Get all the listings's name, price, picture_url , coordinates in 1km radius. (geopspatial index)**

The user searches for locations within a kilometer radius around him. The application shows all the listings on a map. This would require the coordinates of the location, along with a few other properties as name and price(projections).
To achieve this a **geospatial index** is applied on coords attribute of location in listings documents.

```
mongos>  db.listings.createIndex( { "location.coords": "2dsphere" }
);
{
        "raw" : {

"rs1/it.unige.dibris.lsccluster.node6:27017,it.unige.dibris.lsccluster.node7:27017
" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 4,
                        "numIndexesAfter" : 5,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                },

"rs2/it.unige.dibris.lsccluster.node8:27017,it.unige.dibris.lsccluster.node9:27017
" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 4,
                        "numIndexesAfter" : 5,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                },

"rs0/it.unige.dibris.lsccluster.node3:27017,it.unige.dibris.lsccluster.node4:27017
,it.unige.dibris.lsccluster.node5:27017" : {
                        "createdCollectionAutomatically" : false,
                        "numIndexesBefore" : 4,
                        "numIndexesAfter" : 5,
                        "commitQuorum" : "votingMembers",
                        "ok" : 1
                }
        },
        "ok" : 1,
        "operationTime" : Timestamp(1631380582, 7),
        "$clusterTime" : {
                "clusterTime" : Timestamp(1631380582, 7),
                "signature" : {
                        "hash" : BinData(0,"KaIuIyJOXKkrp+nNmoCfxFpFrE0="),
                        "keyId" : NumberLong("6975140761171918849")
                }
        }
}
```

```
db.listings.find( { 'location.coords': { $geoWithin: { $centerSphere: [ [ 52.3663823,4.9042306] , 1 / 6378.1 ] } } },
    { "name": 1 , "location.coords" : 1, price : 1});
```


```
{ "_id" : 48638609, "name" : "Ideal Double Standard At Amsterdam", "price" : 154, "location" : { "coords" : [ 52.3703, 4.89622 ] } }
{ "_id" : 10496067, "name" : "city cultural apartment Amsterdam", "price" : 98, "location" : { "coords" : [ 52.36984, 4.89629 ] } }
{ "_id" : 42417593, "name" : "Spacious townhouse in centre Amsterdam", "price" : 90, "location" : { "coords" : [ 52.36888, 4.89635 ] } }
{ "_id" : 16825279, "name" : "Best view in the hart of the center", "price" : 200, "location" : { "coords" : [ 52.36891, 4.89586 ] } }
{ "_id" : 809026, "name" : "Boss", "price" : 119, "location" : { "coords" : [ 52.36925, 4.89579 ] } }
{ "_id" : 46348037, "name" : "Rembrandt Studio One", "price" : 45, "location" : { "coords" : [ 52.36722, 4.89531 ] } }
{ "_id" : 8303992, "name" : "The Secret Chapel", "price" : 400, "location" : { "coords" : [ 52.36597, 4.89574 ] } }
{ "_id" : 19030625, "name" : "Apartment next to Dam Square | With a cat", "price" : 225, "location" : { "coords" : [ 52.37175, 4.89931 ] } }
{ "_id" : 9947853, "name" : "Spacious appartment in city centre", "price" : 185, "location" : { "coords" : [ 52.37151, 4.89919 ] } }
{ "_id" : 12971906, "name" : "Authentic Amsterdam Canal Apt WITH PRIVATE CHEF", "price" : 57, "location" : { "coords" : [ 52.3695, 4.89661 ] } }
{ "_id" : 46939988, "name" : "Oudezijds Voorburgwal - 1 bedroom - Sleeps 2", "price" : 90, "location" : { "coords" : [ 52.37045, 4.89679 ] } }
{ "_id" : 28798431, "name" : "Delft Blue Suite", "price" : 25, "location" : { "coords" : [ 52.37021, 4.89717 ] } }
{ "_id" : 8413979, "name" : "Comlete independent quiet studio", "price" : 36, "location" : { "coords" : [ 52.37059, 4.89758 ] } }
{ "_id" : 48660090, "name" : "Exclusive Double Premium At Amsterdam", "price" : 226, "location" : { "coords" : [ 52.37054, 4.89769 ] } }
{ "_id" : 20405959, "name" : "Renovated, Cozy Room in City Centre", "price" : 25, "location" : { "coords" : [ 52.37037, 4.89785 ] } }
{ "_id" : 23119390, "name" : "Perfect Canal apartment @RedLightDistrict", "price" : 90, "location" : { "coords" : [ 52.37113, 4.89846 ] } }
{ "_id" : 5675874, "name" : "Beautiful apartment with great view", "price" : 200, "location" : { "coords" : [ 52.37161, 4.89862 ] } }
{ "_id" : 34052508, "name" : "Double Bed, Single Room in the Heart of Amsterdam", "price" : 145, "location" : { "coords" : [ 52.37184, 4.89854 ] } }
{ "_id" : 47012402, "name" : "Longterm centre Amsterdam", "price" : 111, "location" : { "coords" : [ 52.37193, 4.89813 ] } }
{ "_id" : 251444, "name" : "Luxurious apartment at the Prinsengracht", "price" : 65, "location" : { "coords" : [ 52.37169, 4.89807 ] } }
```


10. Get the number of properties grouped by property type.($sum)

```
db.listings_1.aggregate(
      [
       {
        "$group":
         {
          "_id": "$property_type",
          "count": { '$sum': 1 },
```

```
        }
      }
    ]
  );
{ "_id" : "Entire condominium", "count" : 251 }
{ "_id" : "Shared room in houseboat", "count" : 2 }
{ "_id" : "Private room in cabin", "count" : 3 }
{ "_id" : "Entire cabin", "count" : 5 }
{ "_id" : "Room in hotel", "count" : 101 }
{ "_id" : "Entire guesthouse", "count" : 16 }
{ "_id" : "Bus", "count" : 1 }
{ "_id" : "Private room in apartment", "count" : 2152 }
{ "_id" : "Tipi", "count" : 1 }
{ "_id" : "Entire cottage", "count" : 12 }
{ "_id" : "Private room in guest suite", "count" : 106 }
{ "_id" : "Entire place", "count" : 8 }
{ "_id" : "Shared room in boat", "count" : 5 }
{ "_id" : "Campsite", "count" : 2 }
{ "_id" : "Houseboat", "count" : 186 }
{ "_id" : "Private room in lighthouse", "count" : 1 }
{ "_id" : "Entire home/apt", "count" : 2 }
{ "_id" : "Private room in tiny house", "count" : 9 }
{ "_id" : "Private room in hostel", "count" : 26 }
{ "_id" : "Private room in townhouse", "count" : 167 }
```

11. Add a listing (User Becomes a Host by listing his property)

This workload requires 2 collections to be updated (**multi document transaction**).
The user becomes a host, which means that is_host, host_since, host_is_superhost, host_identity_verified are to be added in the user's document.
Simultaneously, the user's first listing is added to the listing's collection.

Such transactions don't occur as often in this application because once the user becomes a host, his next listing would only involve a single insert to the listings collection (one collection only) as is_host, host_since, host_is_superhost, host_identity_verified properties have already been set. If such scenarios do occur, to provide atomicity for multiple documents, Transactions are used.

Query to Add a listing for a new host.

```
var session1 = db.getMongo().startSession({readPreference: { mode:
"primary" }});
var sessionUser =
session1.getDatabase('projectUser2').getCollection('users_bookings');
var sessionListing =
session1.getDatabase('projectUser2').getCollection('listings_1');
var newListingID = 340992;
```

```
var location = {country:"NL",
city:"Amsterdam",neighbourhood_area:"Centrum-Oost",coords:[52.37,
3.89393]};
var amenities = ["wifi", "kitchen"];
session1.startTransaction({readConcern:{level: 'snapshot'},writeConcern:
{w: 'majority'}});
var host_date = new Date(2021, 09, 09, 13, 12);

try {
sessionUser.update(
{_id: 33284},
{'$set' : {is_host: true, host_since: host_date, host_is_superhost: 0,
host_identity_verified : 1}
});
sessionListing.insertOne(
{_id: 340992, user_id : 33284, location:location, name : "Stylish  light
apartment in de pijp",description: "My stylish and comfortable one bedroom
apartment is located in the most beautiful place in
Amsterdam.",neighborhood_overview:"The Pijp is one of most enjoyable areas
of Amsterdam. It is very peaceful and
beautiful.",picture_url:"https://a0.muscache.com/pictures/ba15946c-c019-
46de-85b6-
3d83bd896a7a.jpg",price:70,has_availability:true,instantly_bookable:false,
property_type:"Entire apartment",room_type:"Entire
home/apt",accommodates:2,bathrooms_text:"1 bath",bedrooms:1,beds:1,
amenities: amenities
});
} catch (error) {
session1.abortTransaction();
throw error;
}
session1.commitTransaction();
session1.endSession();

{ "acknowledged" : true, "insertedId" : 340992 }
```

Query to Add a listing for a user that is already a host.

This query does not need transactions since only a single document is being updated

```
mongos> var location =
{country:"NL",city:"Amsterdam",neighbourhood_area:"Zuid",coords:[52.339657
6,4.8711542]};
mongos> var amenities = ["wifi", "kitchen"];
mongos> var host_date = new Date(2021, 09, 10, 1, 12);
mongos> db.listings_1.insertOne({_id: 340993, user_id : 33284,
location:location, name : "Stylish  light apartment in Zuid",description:
"My stylish and comfortable one bedroom apartment is located in the most
beautiful place in Amsterdam, close to the
subway.",neighborhood_overview:"The Zuid is one of most enjoyable areas of
Amsterdam.",picture_url:"https://a0.muscache.com/pictures/ba15946c-c019-
46de-85b6-
```

```
3d83bd896a7a.jpg",price:120,has_availability:true,instantly_bookable:true,
property_type:"Entire apartment",room_type:"Entire
home/apt",accommodates:4,bathrooms_text:"2 bath",bedrooms:3,beds:4,
amenities: amenities});

{ "acknowledged" : true, "insertedId" : 340993 }
```

12. User writes a review about one listing (with all review_scores_rating,review_scores_cleanliness, review_scores_checkin, review_scores_communication, review_score_value, review_scores_location).

```
db.reviews.insertOne({
    _id:851025,
    reviewer_id:1008593,
    comments:"Excellent accommodation, close to everything,
    lovely hosts, beautiful",date:
    new Date("2021-09-12T22:00:00.000+00:00"),
    review_scores_rating:99,
    review_scores_accuracy:10,
    review_scores_cleanliness:10,
    review_scores_checkin:10,
    review_scores_communication:10,
    review_scores_location:10,
    review_scores_value:10,
    listing_id:340992
});

{ "acknowledged" : true, "insertedId" : 851025 }
```

13. Get all reviews given a listing ID.

```
mongos> db.reviews_1.find({listing_id : 340992 }).pretty();
{
        "_id" : 851025,
        "reviewer_id" : 1008593,
        "comments" : "Excellent accommodation, close to everything,
lovely hosts, beautiful",
        "date" : ISODate("2021-09-12T22:00:00Z"),
        "review_scores_rating" : 99,
        "review_scores_accuracy" : 10,
        "review_scores_cleanliness" : 10,
        "review_scores_checkin" : 10,
        "review_scores_communication" : 10,
        "review_scores_location" : 10,
        "review_scores_value" : 10,
        "listing_id" : 340992
}
```

14. Get the average rating of each property based on user reviews (cleanliness, checkin, communication, value, accuracy)

For this query, partition schema does not matter as the entire collection has to be queried.

```
db.reviews_1.aggregate(
            [
              {
                "$group":
                 {
                   "_id": "$listing_id",
                   "avgRatingOverall": { "$avg": "$review_scores_rating"
},
                   "avgRatingAccuracy": { "$avg": "$review_scores_accurac
y" },
                   "avgRatingCleanliness": { "$avg": "$review_scores_clea
nliness" },
                   "avgRatingCheckin": { "$avg": "$review_scores_checkin"
 },
                   "avgRatingCommu": { "$avg": "$review_scores_communicat
ion" },
                    "avgRatingValue": { "$avg": "$review_scores_value" }
                 }
              }
            ]
         );
{ "_id" : 19782977, "avgRatingOverall" : 96, "avgRatingAccuracy" : 10,
"avgRatingCleanliness" : 10, "avgRatingCheckin" : 10, "avgRatingCommu" : 10,
"avgRatingLocation" : 10, "avgRatingValue" : 9 }
{ "_id" : 19940438, "avgRatingOverall" : 100, "avgRatingAccuracy" : 10,
"avgRatingCleanliness" : 10, "avgRatingCheckin" : 10, "avgRatingCommu" : 10,
"avgRatingLocation" : 10, "avgRatingValue" : 10 }
{ "_id" : 41614333, "avgRatingOverall" : 90, "avgRatingAccuracy" : 9,
"avgRatingCleanliness" : 10, "avgRatingCheckin" : 10, "avgRatingCommu" : 10,
"avgRatingLocation" : 10, "avgRatingValue" : 9 }
{ "_id" : 7299468, "avgRatingOverall" : 100, "avgRatingAccuracy" : 10,
"avgRatingCleanliness" : 10, "avgRatingCheckin" : 10, "avgRatingCommu" : 10,
"avgRatingLocation" : 10, "avgRatingValue" : 10 }
{ "_id" : 7619029, "avgRatingOverall" : 100, "avgRatingAccuracy" : 10,
"avgRatingCleanliness" : 10, "avgRatingCheckin" : 10, "avgRatingCommu" : 10,
"avgRatingLocation" : 10, "avgRatingValue" : 10 }
{ "_id" : 11317998, "avgRatingOverall" : 96, "avgRatingAccuracy" : 10,
"avgRatingCleanliness" : 10, "avgRatingCheckin" : 10, "avgRatingCommu" : 10,
"avgRatingLocation" : 9, "avgRatingValue" : 10 }
{ "_id" : 2545716, "avgRatingOverall" : 84, "avgRatingAccuracy" : 7,
"avgRatingCleanliness" : 8, "avgRatingCheckin" : 9, "avgRatingCommu" : 8,
"avgRatingLocation" : 8, "avgRatingValue" : 8 }
.. so on
```

15. Order the average rating of neighbourhood area based on user score of the location.

```
db.reviews.aggregate(
            [
              {
                "$group":
```

```
                {
                  "_id": "$neighbourhood_area",
                  "avgRatingLocation": { "$avg": "$review_scores_locatio
n" },
                }
              },
              {"$sort":{avgRatingLocation : -1}}
            ]
          );
```

```
{ "_id" : "Centrum-West", "avgRatingLocation" : 9.969346057611306 }
{ "_id" : "Centrum-Oost", "avgRatingLocation" : 9.903504626895058 }
{ "_id" : "De Pijp - Rivierenbuurt", "avgRatingLocation" : 9.825147144159828 }
{ "_id" : "Zuid", "avgRatingLocation" : 9.682558096022154 }
{ "_id" : "De Baarsjes - Oud-West", "avgRatingLocation" : 9.6628544697874 }
{ "_id" : "Westerpark", "avgRatingLocation" : 9.561889433693903 }
{ "_id" : "Oud-Oost", "avgRatingLocation" : 9.522287718614507 }
{ "_id" : "Oud-Noord", "avgRatingLocation" : 9.472269906245874 }
{ "_id" : "Oostelijk Havengebied - Indische Buurt", "avgRatingLocation" :
9.421457647911671 }
{ "_id" : "Noord-Oost", "avgRatingLocation" : 9.389780769894701 }
{ "_id" : "IJburg - Zeeburgereiland", "avgRatingLocation" : 9.336862911195821 }
{ "_id" : "Osdorp", "avgRatingLocation" : 9.31387132763109 }
{ "_id" : "Buitenveldert - Zuidas", "avgRatingLocation" : 9.295803183791607 }
{ "_id" : "Watergraafsmeer", "avgRatingLocation" : 9.251362950933766 }
{ "_id" : "Slotervaart", "avgRatingLocation" : 9.192068334350214 }
{ "_id" : "De Aker - Nieuw Sloten", "avgRatingLocation" : 9.168248090719741 }
{ "_id" : "Bos en Lommer", "avgRatingLocation" : 9.122705143996715 }
{ "_id" : "Bijlmer-Centrum", "avgRatingLocation" : 9.115959719255416 }
{ "_id" : "Noord-West", "avgRatingLocation" : 9.067109482363719 }
{ "_id" : "Geuzenveld - Slotermeer", "avgRatingLocation" : 9.062358276643991 }
```

16. Get the average rating of the neighbourhood area “Zuid”. ($match)

```
db.reviews.aggregate(
          [
                  { "$match" : { neighbourhood_area : "Zuid" } },
                {
                  "$group":
                    {
                      "_id": "$neighbourhood_area",
                      "avgRatingLocation": { "$avg": "$review_scores_location" }
   ,
                    }
                } ]);
```

17. Average price of Zuid and Osdorp areas neighbourhood. (map reduce and $in)

With the help of map reduce, the aggregate can be stored as a collection and it can be queried like a normal collection from time to time. This approach can be use

```
var mapFunction1 = function() {
    emit(this.location.neighbourhood_area, this.price);
  };
```

```
var reduceFunction1 = function(keyCustId, valuesPrices){
  return Array.avg(valuesPrices);
};
db.listings.mapReduce(
  mapFunction1,
  reduceFunction1,
  { out: "map_reduce_location_price" }
);
db.map_reduce_location_price.find( { "_id": { "$in": [ "Osdorp", "Zuid" ] }});
```

```
{ "_id" : "Osdorp", "value" : 108.5 }
{ "_id" : "Zuid", "value" : 167.1990915972748 }
```

```
mongos> show collections;
listings
listings_1
map_reduce_location_price
reviews
reviews_1
users_bookings
```

Observations:

$lookup can only be performed on unsharded collections. We tried to avoid joins as they require high data communication and are expensive. Any query that requires a join can be done at the application level.