CIS 550 Database

Yecheng Yang, Peng Yan, Hugang Yu, Shangcheng Jiang

EAT & CHILL

A Web Suggestion Application for Hotels and Restaurants

Introduction

The main idea behind Eat & Chill is to create a web suggestion application for hotels and nearby restaurants based on user input, whether the input is hotel name, city name or just a zip code.

Our motivation for this project is that when people are on trips for different purposes and travel to a new city, they may have a hard time to find specific restaurants near the hotel. For example, one may wish to have a restaurant that will cater to his/her need during the trip and is unwilling to waste time on getting to the restaurants multiple times a day. This is where our application will be helpful. Since it is not convenient to look at a hotel booking

application (such as Expedia) and another application for restaurant reviews (such as yelp) at the same time, our application tries to create a smooth user experience by combining them. We aim to integrate information from both ends, render a simultaneous view and facilitate informed decisions.

Data Set and Cleaning

We used two sources of data: one is from Expedia Affiliate Network's hotel property data -- a list of all active properties in the EAN database in English text; the other is from Yelp database which contains information about restaurants. Here is all the information we can get from the data source:

- Properties can be sorted by performance via the Sequence Number field.
    - The Sequence Number value ranks properties based on EAN transactional data from the last 30 days, with one indicating the best-performing hotel and others following in ascending numerical order.
    - The same sorting algorithm is the same as the one used for dateless hotel list requests; all values are updated daily.
- The Location field is a short description of the location of the property. A possible value for this field is "Near Times Square."
- Latitude/Longitude of properties can be joined with other restaurant sources to provide geographical information.
- Hotel zip code can be used to join with restaurant zip code to provide nearby restaurant recommendation.
- City field contains most major cities in the U.S, which can also be utilized to query city related questions.
- Hotel star rating is measured in a range from one to five

Another data source is restaurant data from yelp open data source. This is a well-formed one, and we can deal with it with ease.

We made a local copy of MySql Server to perform all operations and then we use mysqldump command to export data in the server and then import to remote server,

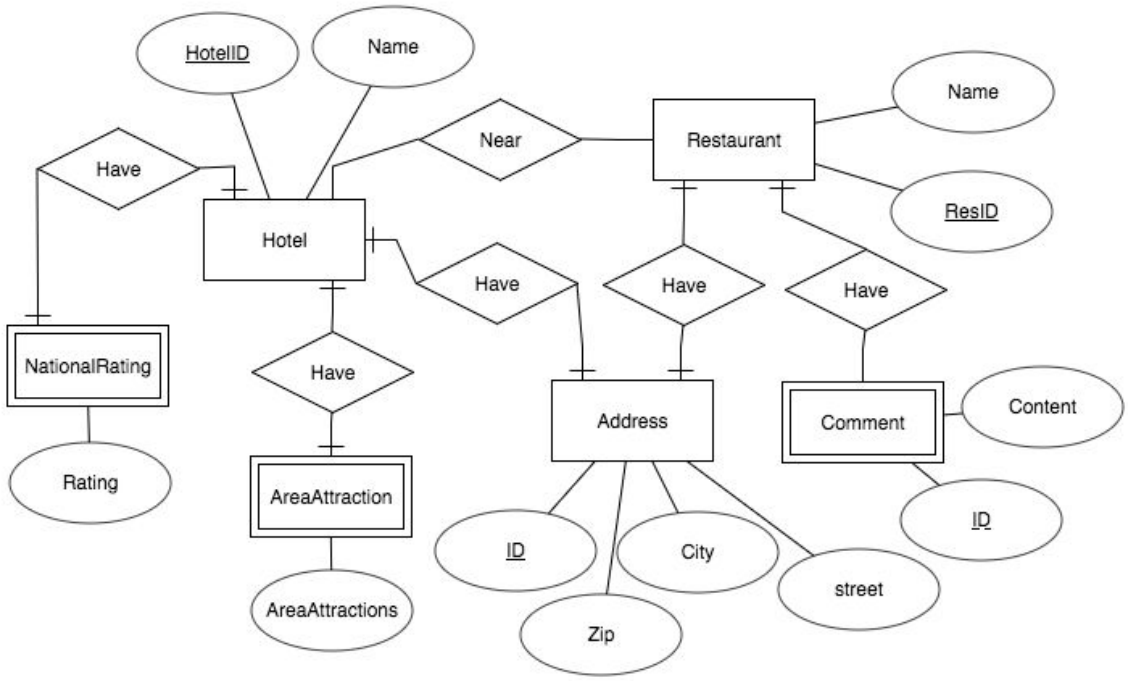
which saved us a lot of time.

Finally, when constructing data tables, we specify the foreign key and not null constraints on some columns, which saved us a lot of cleaning time, because it is handled by SQL server itself.

Database Usage

MySQL Design and Usage

We used MySQL server deployed on Amazon AWS to provide service. All data except for restaurant comments are stored on MySQL server. We have created an index on all primary keys and foreign keys to speed up the query. Further optimizations will be discussed later in the report. The detailed design of our schema lies below.



There are five tables related to Hotel information, and there are seven tables in yelp restaurant database. To combine the two sources, we used Zip code and City name to join

the two kinds of properties. We obtain the hotel tables by decomposing the hotel data source into five tables: 1. Hotel raw information, 2. Area Description, 3. Dining Description, 4. SPADescription, 5. Hotel star rating. The primary key of the original dataset is hotel id, and we keep this information in each of the new tables so that information and tables can be joined if necessary. By doing so, we achieve a BCNF relation as the only non-trivial dependencies are those in which a key determines some attributes. Therefore, each tuple can be thought of as an entity or relationship, identified by the remaining attributes.

We decided to have most of the queries performed on MySQL server since some of them might potentially get complicated. And MySQL database provides us with the ability do execute such queries as opposed to NoSQL database.

MongoDB Usage

On top of MySQL database, we also decided to apply MongoDB as a complimentary database to handle the comments for the restaurants in our data set. There a few key factors that facilitated this decision:

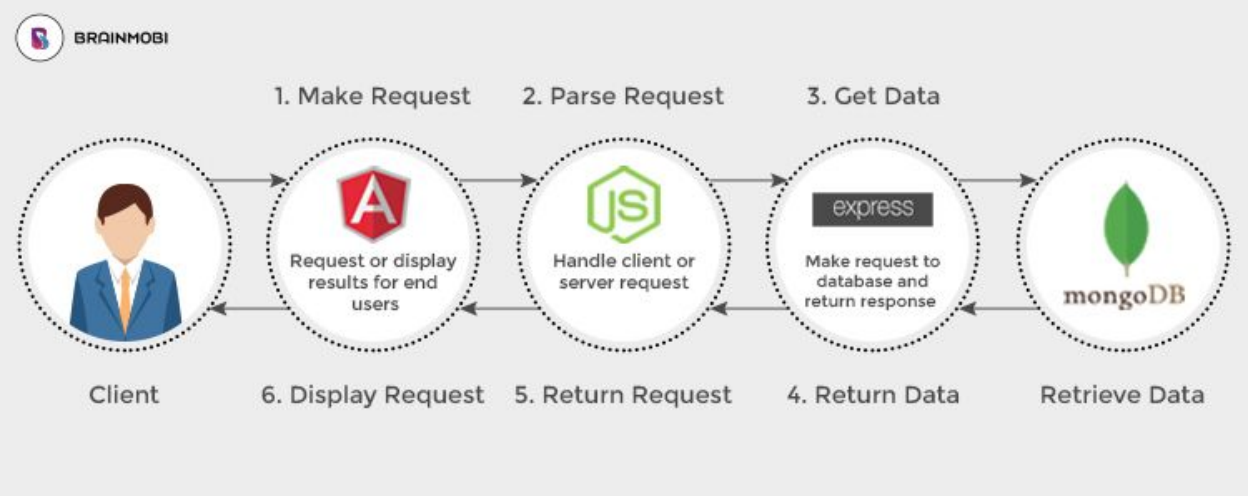
- Large amount of data for comments: there are roughly 4.8 million rows for the comments data set. Altogether, those files take up to 3.8 gigabytes on disk.
- JSON format: dataset came in JSON format, which is suitable for MongoDB.
- Schemaless: dataset for comments does not have a well formed schema and they are subject to future updates from the users.
- Simple Query: only key-value match needed.

With most of our dataset staying on MySQL database, we want a convenient way to connect two databases. We applied an approach to combine them with business id as our feature aims to provide reviews for each restaurant separately. Business id is used over business name (restaurant name) for two main reasons. There might be duplicates in business name but not in business id. Also, business name is not contained in the comment dataset, and string matching through MySQL gives us more freedom (in case we want to check restaurants with names) and can ensure MongoDB is strictly key value matching. To do so, we first query MySQL for identifying business id based on business name, and they query MongoDB based on returned business id.

Architecture

For the framework, we use angular in the front end, express and node in the backend, pretty much similar to that of the homework, because we reused that code in the beginning. Later we made a lot of improvements to make it more modularized so that each of us can efficiently put the right code in the right place with only necessary communication. For example, we created a file to include queries, connection with the database and connection timeouts. We only export some interfaces that express module can make calls to. This approach separates the work between the frontend and backend well.

The architecture graph of our system can be viewed as follows.



Features

On our website, we implemented six features including:

1) Find restaurants near a hotel
2) Find the most popular restaurants near a hotel
3) Find top rated areas
4) Find top hotels for foodaholic
5) Find attractions near a hotels

6) View restaurant customer comments

For each feature, we created a page object for making query requests and displaying the results in a HTML table.

Find the Most Popular Restaurants near a Hotel

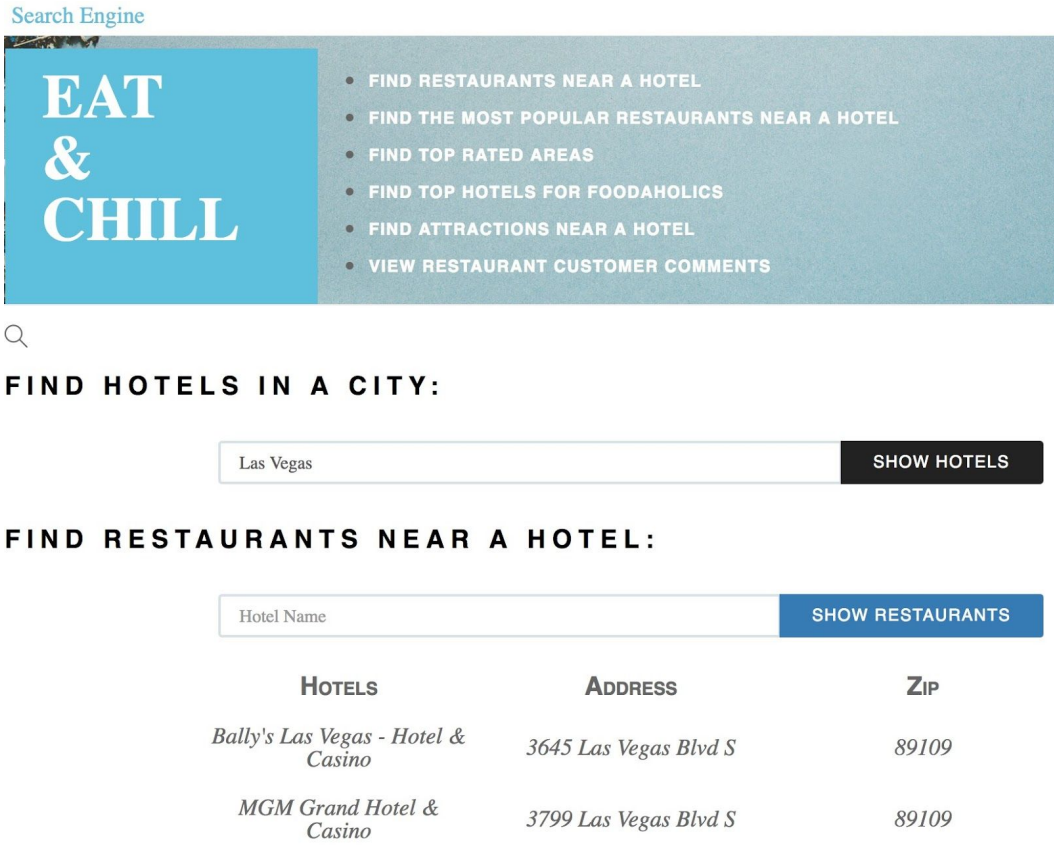
With this feature, users can quickly check the restaurants nearby sorted by their popularity index calculated on the total number of comments users gave to the restaurant. The more comments a restaurant received, the higher it's ranked. With popularity index, the returning results are more informatic than the previous query result.

```
select R.name as name, R.address as address, R.review_count/100 as count
from yelp_db.business R, db550.Hotel H
where R.postal_code = H.Zip and H.name = "' + hotel + '"'
order by count DESC\n' +
limit 10;
```

Find Restaurants near a Hotel

The two most basic but useful queries are on this page: one is for finding the information about all the hotels in a city, including Hotel Name, Address, and Zip Code; another is for find the information about all the restaurants near a hotel, including Restaurant Name, Address, and Zip Code. Having these two queries, we can easily get hotel/restaurant names for making other query requests.

```
select select h.Name, h.Zip, h.address
from db550.Hotel h
where H.City="' + name + '"

select *
from yelp_db.business r, db550.Hotel h
where r.postal_code = h.Zip and h.name= "' + name + '"
```



Find Top Rated Areas

The feature integrates the information of hotel, restaurants, and Zip altogether to provide users with a smoother web interaction experience. There are three queries on this page:

a) Find a list of Zip sorted by the rate of this area. The rate of an area is calculated on the average rate of the restaurants combined with the average rate of the hotels in this area.
b) Find the top hotels in an area.
c) Find the top restaurants in an area.

To exclude the corner case in which an area with only one 5-star hotel and one 5-star restaurant are returned, the query also specifies that only areas with at least five restaurants and three hotels are qualified to be considered as candidates of top areas.

```
select rr.rzip as zipcode, TRUNCATE((rr.rstar + hr.hstar),2) as rate
from (select r.postal_code as rzip, avg(r.stars) as rstar
    from yelp_db.business r
    group by r.postal_code
    having count(*) >= 5) as rr,
   (select h.Zip as hzip, avg(s.StarRating) as hstar
    from db550.Hotel h, db550.Star s
    where h.EANHotelID = s.HotelID and h.City = 'Las Vegas'
    group by h.Zip
    having count(*) >= 3) as hr
where rr.rzip = hr.hzip
order by rate DESC
```

Find Top Hotels for Foodaholic

Other than restaurants information, our website also provides users with hotel information. The query calculates the average rate of all the restaurants near a hotel, associates the rate with the hotel, and then ranks the hotels in a city based on the rate in descending order. The higher the rate, the higher the restaurants near a hotel are rated.

```
select Select H2.name as name, H2.Address as address,
convert(averageStar, DECIMAL(5,2)) as averageStar
from (select H.EANHotelID, H.Address, H.name, R.id, avg(R.stars) as averageStar
from db550.Hotel H, yelp_db.business R where H.Zip = R.postal_code
and H.city = 'Las Vegas' Group by H.EANHotelID) as H2
order by averageStar DESC
limit 10
```

Find Attractions near a Hotel

The feature is rather straightforward. The query uses hotel name as the primary key to return the attraction description associated with the hotel.

```
select h.name as name, a.Description as Description
from db550.Hotel h, db550.AreaAttractions a
where h.EANHotelID = a.HotelID and h.Name = "' + hotel +'";
```

View Restaurant Customer Comments

Users can directly check the comments of a restaurant after they log in. We used MongoDB to as the framework of the database for this part. Detailed elaboration about why we chose a non-relational database is further elaborated in the MongoDB section.

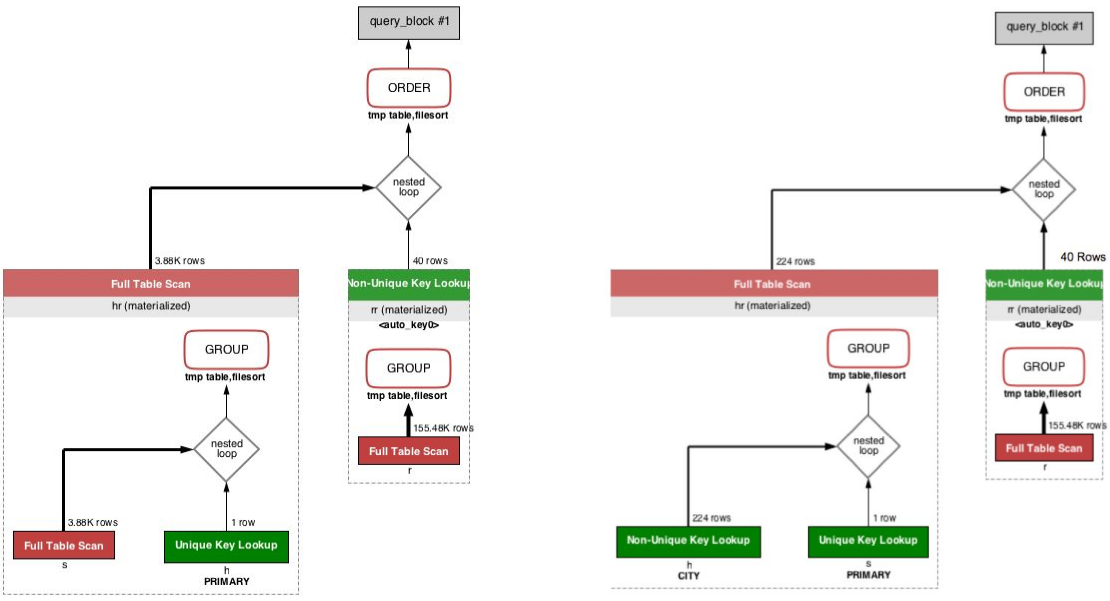
```
var query = 'select r.id from yelp_db.business r where r.name = "'+ name +'"';

var pro = ndb.collection('comments').find({business_id:id}).limit(100);
```

Query Optimization

Query optimization is also an essential aspect of the project for a few reasons. First of all, if the queries are not scalable, then database performance will deteriorate with an increase in database size. This is a valid concern as we expect more businesses as well as more reviews to be added. Therefore, we paid special attention to ensure the database performance is optimized and scalable.

MySQL Optimization

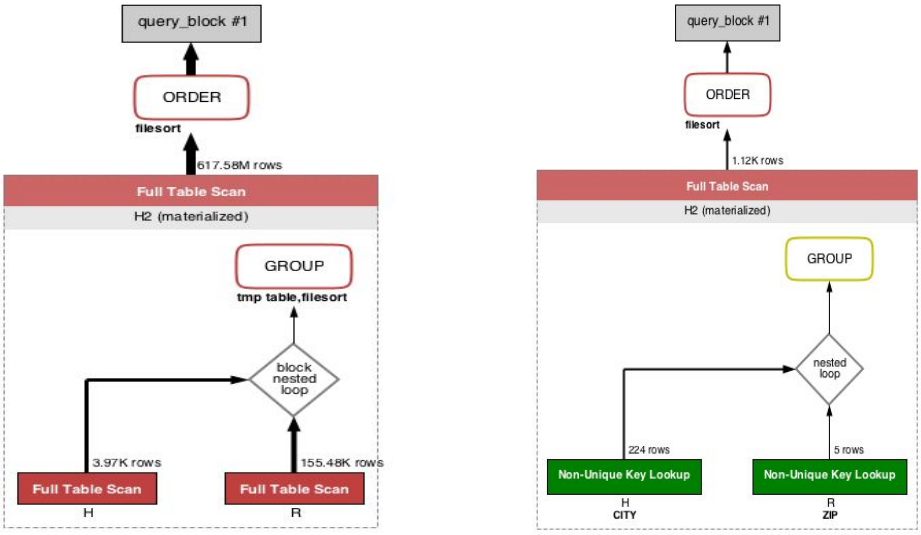
As for MySQL database, the primary factor for performance lies in joining multiple tables for complex queries mentioned earlier. To improve the efficiency of query execution, we



added an index on some of the frequently used fields which were used as the condition in joining in our supported queries. And we achieve a significant reduction in the size of joined tables through this implementation.

Below is the execution plan of our most complex query (feature three above). It joins views created by two subqueries, one of which includes joining two more tables.

We can see that the query execution plan of the original schema requires full table scan of table s (table on bottom left) as well as one for table r. They roughly require scanning 4,000 and 155,480 rows. After nested join, subquery on the left will render a view of 4,000 rows. After optimization, query processor can now utilize non-unique key lookup for constant time access on fields of the table, and there are only 200 rows as the output of the subquery. It is important to note that the subquery on the right still require a full table scan since the query involves aggregation and grouping later on.



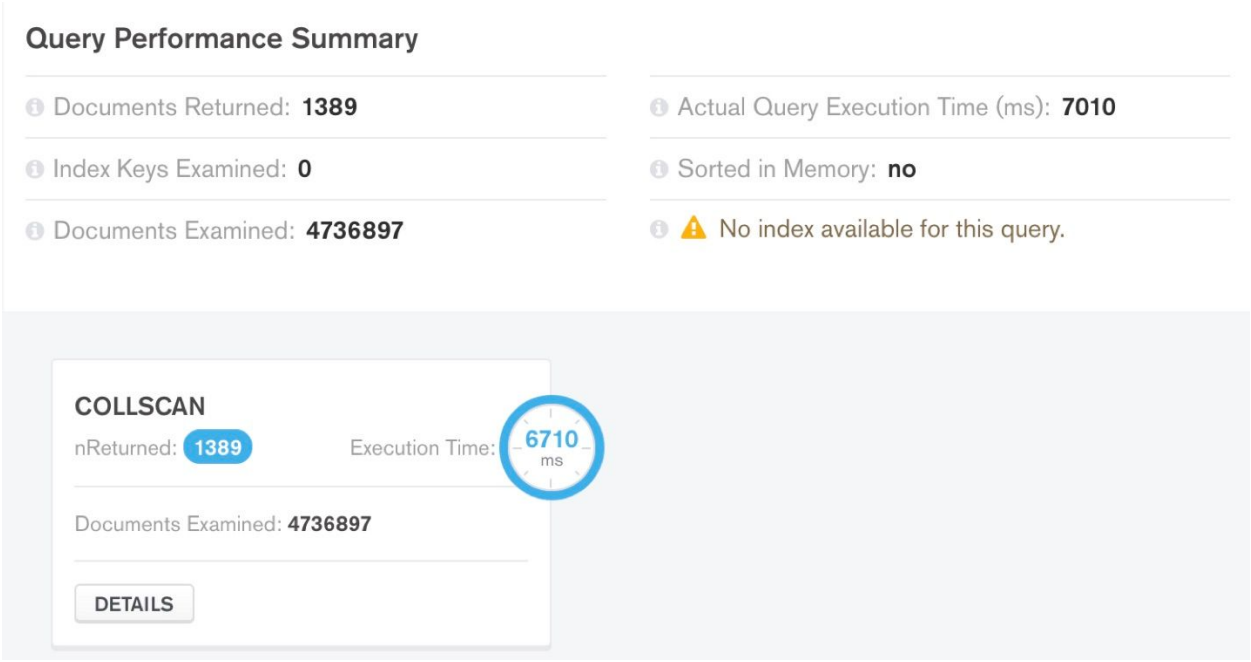
Here is another example of query optimization (feature four above). We can see that query involves aggregation and a subquery, in which two tables are joined. Before optimization, processor performs a full table scan on both table H and table R, which will scan 4,000 and 155,480 rows respectively.

This will lead to a massive table as the result of the subquery. As we can see from the execution plan, the joined table contains 617 million rows, and the processor will sort all those rows. However, after applying the index to the columns, query processor can now merely perform non-unique key lookup. It will only select 200 rows from table H and five rows from table R, and the result of the subquery is just 1000 rows. In this case alone, indexing reduces the amount of required work by several orders of magnitude.

MongoDB Optimization

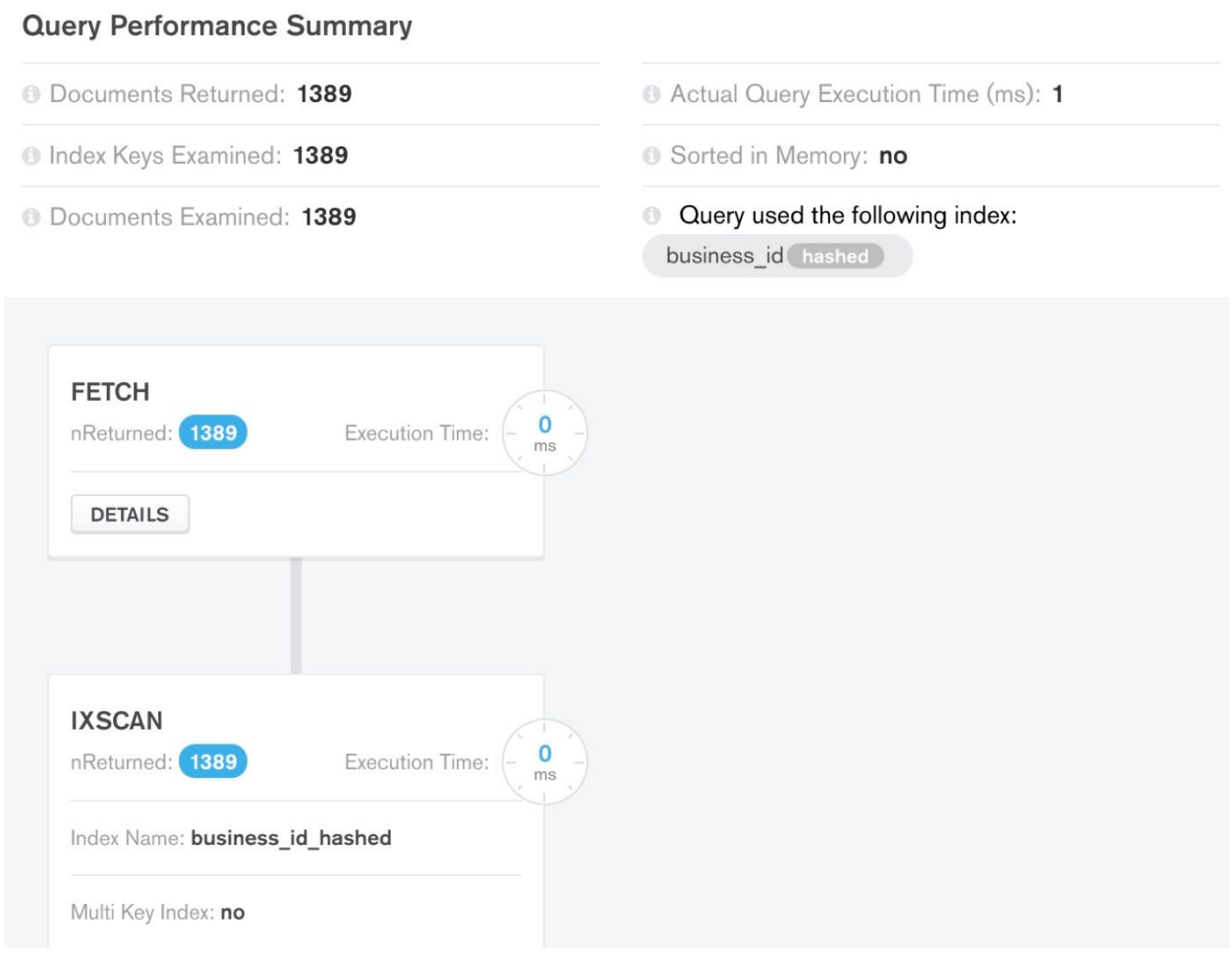
In case of MongoDB, as mentioned above, it is responsible for searching comments for given restaurants. Therefore, the major factor that limits performance lies in searching the comments table. To improve the efficiency of query execution, we added an index on the field which is used for key-value lookup.

Below is the execution plan for one example query. There are a few places that should be noticed. First of all, query processor examined all 5 million rows in the table. Query execution time is 7,010 milliseconds.



The below picture is the execution plan of the same query with hashed index on business id. Now, number of documents examines is reduced to 1389 rows, which is exactly the same as the number of rows to be returned. Furthermore, actual query execution time is just 1 millisecond now.

This is extremely important because the result of optimization is not specific to one single query, but instead, this improvement can be applied to every query to be run on MongoDB. Under the original schema, every query demands a full table scan of 5 million rows and takes 7,000 milliseconds. After optimization, every query directly access documents to be returned directly and runtime of each query reduces from 7,000 milliseconds to just 1 millisecond. Overall, this optimization provides us 7000x in runtime improvement for every single query.



Bonus Features

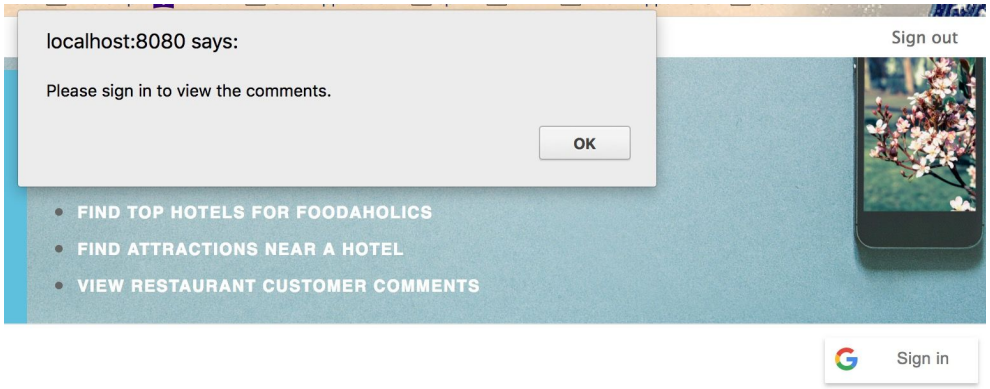
Google Login Authentication

We integrated Google login API into our web page, and users can login on any feature page.

```
<script src="https://apis.google.com/js/platform.js" async defer></script>
<meta name="google-signin-client_id"
content="171822122274-onse4r15fpj0u7r2fqb3rklpop9u2kek.apps.googleusercontent.com">

 <span class="g-signin2" data-onsuccess="onSignIn" ></span>

<a href="#" onclick="signOut();" id="sign-out" >Sign out</a>
```

Access Control

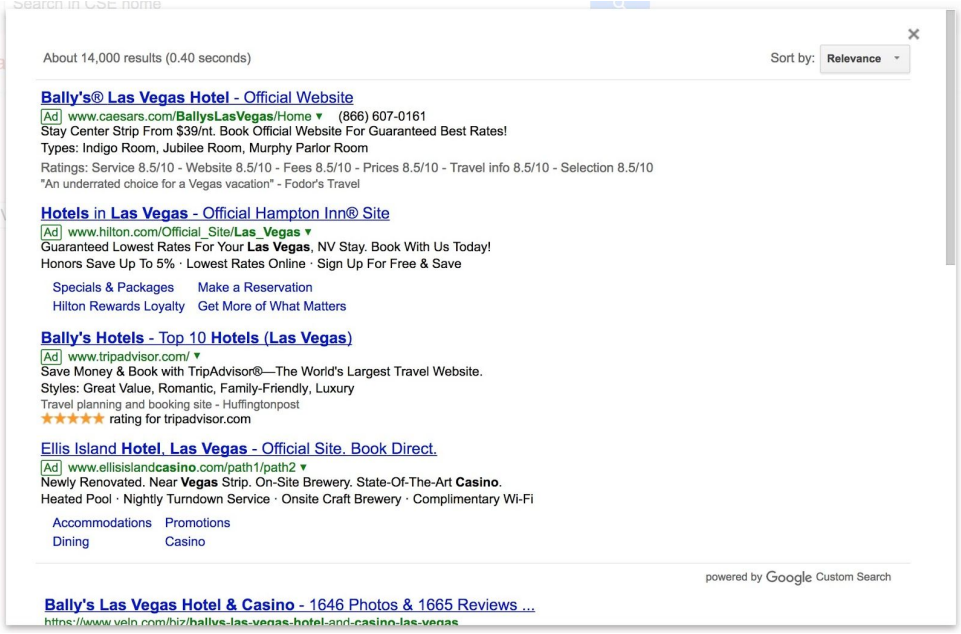
We also utilized login feature further by associate access control with users' login status. Only users who have signed in see the comments of a restaurant.



Search through Google

We also integrated a Customized Google search on our webpage. Users who want to get more detailed information about a hotel/restaurant can use the 'Search Engine' to view

more relevant information. This customized search engine only search results on Yelp.com and expedia.com to ensure the results are the most relevant.



Technical Difficulties

- Data cleaning: The raw hotel data we get is a whole big text file and we need to clean it into several well formed csv files. Please refer to data cleaning section to section how we tackled this difficulty.
- Populate MySql server data: 1. Choose the right schema so that the database can maintain a consistent state. 2. Directly pushing to the remote server can be too slow for us to tolerate.Please refer to data population section to section how we tackled this difficulty.
- MEAN Stack: How to correctly combine and use Angular, Express and Node.js and create modules for each of us to work on with only necessary communication.
- Front-end Design: How to integrate HTML, Angular.js, and database request altogether and show the results on the web page in an informative and intuitive way. We used Bootstrap, HTML, CSS, Angular.js and Node.js to establish the workflow and created a visually pleasant UI design

Future Extensions

Map Integration

One feature that can improve user experience is to integrate map viewing into our web application. With this feature, the users can perform all the aforementioned searches directly on the map, such as to find out the cluster of hotels and restaurants. This will allow a more direct view of the results from our suggestions. The user can see how close the desired restaurants are with respect to the hotels, whether the distance is just one block or a few.

Comments with Pictures

Another feature that can even further enhance user experience is to supply pictures along with the comments for the restaurants. With this feature, users can gain a visual impression of the restaurants and the food they provide.

Clickable Query Results

The current design requires users to copy & paste query result from one page to another to execute a new query. In future development, developers can make the query results clickable and linked to relevant queries so that the web page could be more user friendly and intuitive to use. For example, if a user wants to find a hotel in the city and also check the quality of the restaurants near the hotel, the return hotel could be clickable so that the restaurant information could also show in the same page.