ICCS240: Assignment 2.1

Kriangsak T., Hasdin G.
kriangsak.thi@student.mahidol.edu, hasdin.g@student.mahidol.edu
Collaborator: 14091

1 Yelp

1.1 Report of Schema

This database has four relations:

- BUSINESS(businessId char, name varchar, address varchar, city varchar, state varchar, postalCode varchar, stars float, isOpen numeric(1), reviewcount numeric)

  Primary Key : businessId

- USER(userId char(22), name varchar, reviewCount numeric, averageStars float)

  Primary Key : userId

- ELITEUSER(userIdd char(22), elite INTEGER)

  Foriegn key: userId

  Reference : USERS(userId)

- REVIEW(reviewId char(22), userId char(22), businessId char(22), stars integer, useful integer, text varchar, year integer)

  Primary key: reviewId

  Foriegn key: userId, businessId

Where BUSINESS has customers known as USER where some users can be categorized as ELITE status by any business if certain requirements are met. Also, each of the users can review any business service used.

1.2 Indexes

The following indexes are created as means of efficient queries. We chose to hash the attributes whose likelihood in being queried upon is relatively high.

```
create  index business1 on business using hash(business_id);
create  index business2 on business using hash(address);
create  index business3 on business using hash(city);
create  index business4 on business using hash(stars);
create  index business5 on business using hash(review_count);
create  index business6 on business using hash(is_open);
create  index business7 on business using hash(state);
create  index business8 on business using hash(name);
create  index users1 on users using hash(user_id);
create  index users2 on users using hash(name);
create  index users3 on users using hash(average_stars);
create  index users_elite1 on elite_users using hash(elite);
create  index review1 on review using hash(user_id);
```

```
create  index review2 on review using hash(stars);
create  index review3 on review using hash(text);
create  index review4 on review using hash(business_id);
create  index review5 on review using hash(year);
```

1.3 Queries

(1) Still In Business:

```
SELECT business_id,CONCAT(address,city,state,postal_code) AS
full_address, stars  FROM business  WHERE is_open =1 AND state = state
ORDER BY review_count DESC LIMIT 10;
```

(2) Top Reviews

```
SELECT u.user_id, u.name, review.stars, business_id FROM review
INNER JOIN users u ON business_id= input
ORDER BY review.useful DESC  LIMIT 5;
```

(3) Average Rating

```
SELECT user_id, name AS name_of_user, average_stars AS
average_star_ratings WHERE user_id = id_input;
```

(4) Top Business in City

```
SELECT a.business_id, b.name, b.review_count, stars, countt
FROM (SELECT cit.business_id, COUNT(*) countt
FROM (SELECT business_id, review.user_id
FROM review
   INNER JOIN elite_users answer
   ON answer.user_id = review.user_id AND answer.elite = review.year) el
   INNER JOIN (SELECT * FROM business
   WHERE city = InputCity) cit
   ON cit.business_id = el.business_id
   GROUP BY cit.business_id
   HAVING COUNT(*) > InputeliteCount
   ORDER BY COUNT(*) DESC LIMIT InputTopCount) a
   INNER JOIN (SELECT * FROM business WHERE city = InputCity) b
   ON b.business_id = a.business_id;
```

1.4 Performance Bench-marking

From our experimental results, without indexing, some queries will take a few seconds to output, with indexing, we can run queries within milliseconds. The detailed numerical result is as follows: For

Table 1: Benchmarking queries with and witout indexes

Queries	~indexes	indexes
stillThere	158 ms	74 ms
topReview	33 s 897ms	12 s 267 ms
averageRating	99 ms	16 ms
topBusiness	18 s 239 ms	14 s

Of note, noticed that the first query run after indexing did not perform much better due to heavy-seek searches.