ECE356 Lab 2

Part 2: Yelp query evaluation using SQL explain statement and optimization using index

(a)

Query:

```
SELECT user_id,
       name
FROM user
ORDER BY review_count DESC
LIMIT 1;
```

Explain Result:

```
+----+-------------+-------+------+---------------+------+---------+------+---------+----------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra          |
+----+-------------+-------+------+---------------+------+---------+------+---------+----------------+
|  1 | SIMPLE      | user  | ALL  | NULL          | NULL | NULL    | NULL | 1021667 | Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+---------+----------------+
1 row in set (0.00 sec)
```

From the explain result, we can see that the type is “ALL” and Extra is “Using filesort”, which mean it scan through the entire table and sort it.
In the query, it uses “ORDER BY review_count Desc”. Therefore, we should add an index on the attribute “review_count” in user table.

Add Index:

```
CREATE INDEX review_count_index ON user (review_count) USING BTREE;
```

(b)

Query:

```
SELECT business_id,
       name
FROM business
ORDER BY review_count DESC
LIMIT 1;
```

Explain Result:

```
+----+-------------+----------+------+---------------+------+---------+------+--------+----------------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | rows   | Extra          |
+----+-------------+----------+------+---------------+------+---------+------+--------+----------------+
|  1 | SIMPLE      | business | ALL  | NULL          | NULL | NULL    | NULL | 142527 | Using filesort |
+----+-------------+----------+------+---------------+------+---------+------+--------+----------------+
1 row in set (0.01 sec)
```

From the explain result, we can see that the type is “ALL” and Extra is “Using filesort”, which mean it scan through the entire table and sort it.
In the query, it uses “ORDER BY review_count Desc”. Therefore, we should add index on the attribute “review_count” in business table.

Add Index:

```
CREATE INDEX review_count_index ON business (review_count) USING BTREE;
```

(c)

Query:

```
SELECT avg(review_count) AS avg_review_count
FROM user;
```

Explain Result:

```
+----+-------------+-------+------+---------------+------+---------+------+---------+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------+
|  1 | SIMPLE      | user  | ALL  | NULL          | NULL | NULL    | NULL | 1021667 | NULL  |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------+
1 row in set (0.02 sec)
```

From the explain result, we can see that the type is “ALL”, which mean it scan through the entire table. The query uses “avg(review_count)” which is an aggregation function. Therefore, scanning the entire table is required and no additional explicit indexes are needed.

(d)

Query:

```
SELECT count(*) AS avg_rating_diff_count
FROM user
INNER JOIN
  (SELECT user_id,
          avg(stars) AS avg_rating
   FROM review
   GROUP BY user_id) AS temp ON user.user_id = temp.user_id
WHERE abs(temp.avg_rating - user.average_stars) > 0.5;
```

Explain Result:

```
+----+-------------+------------+--------+---------------+---------+---------+-------------+---------+--------------------------------+
| id | select_type | table      | type   | possible_keys | key     | key_len | ref         | rows    | Extra                          |
+----+-------------+------------+--------+---------------+---------+---------+-------------+---------+--------------------------------+
|  1 | PRIMARY     | <derived2> | ALL    | NULL          | NULL    | NULL    | NULL        | 1655155 | NULL                           |
|  1 | PRIMARY     | user       | eq_ref | PRIMARY       | PRIMARY | 22      | temp.user_id |       1 | Using where                    |
|  2 | DERIVED     | review     | ALL    | NULL          | NULL    | NULL    | NULL        | 1655155 | Using temporary; Using filesort |
+----+-------------+------------+--------+---------------+---------+---------+-------------+---------+--------------------------------+
3 rows in set (0.01 sec)
```

For the outer select count (*) statement, an entire table scan is required.
For the join clause, it is using user_id, which is already a primary key of user table.
For the select statement on review table, it is using aggregation on user_id and calculate avg(stars), which means scanning the entire table is require, no index needed.

(e)

Query:

```
SELECT
count(if(review_count > 10, 1, NULL))/count(*) AS more_than_ten_reviews_fraction
FROM user;
```

Explain Result:

```
+----+-------------+-------+------+---------------+------+---------+------+---------+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------+
|  1 | SIMPLE      | user  | ALL  | NULL          | NULL | NULL    | NULL | 1021667 | NULL  |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------+
1 row in set (0.00 sec)
```

From the explain result, we can see that the type is “ALL”, which mean it scan through the entire table and sort it. In the query, it uses “count(if(review_count > 10, 1, NULL))”. Therefore, we should add index on the attribute “review_count” in user table.

Add index:

```
CREATE INDEX review_count_index ON user (review_count) USING BTREE;
```

(f)

Query:

```
SELECT avg(length(text)) AS avg_review_length
FROM review
INNER JOIN
  (SELECT user_id
  FROM user
  WHERE review_count > 10) AS temp on review.user_id = temp.user_id;
```

Explain Result:

```
+----+-------------+------------+------+---------------+-------------+---------+--------------------+---------+-------------+
| id | select_type | table      | type | possible_keys | key         | key_len | ref                | rows    | Extra       |
+----+-------------+------------+------+---------------+-------------+---------+--------------------+---------+-------------+
|  1 | PRIMARY     | review     | ALL  | NULL          | NULL        | NULL    | NULL               | 1655155 | NULL        |
|  1 | PRIMARY     | <derived2> | ref  | <auto_key0>   | <auto_key0> | 22      | Yelp.review.user_id |      10 | Using index |
|  2 | DERIVED     | user       | ALL  | NULL          | NULL        | NULL    | NULL               | 1021667 | Using where |
+----+-------------+------------+------+---------------+-------------+---------+--------------------+---------+-------------+
3 rows in set (0.01 sec)
```

The query on the table review has type “ALL”, which means it will scan through the entire table.
On the review table, it uses avg function which is an aggregation function, therefore, scanning the entire table is required.
Also, in the join clause, we can see that it is already using the primary key of the review table.
At last, we look at the select statement on the user table. It is obvious that it is scanning the entire table for search for “WHERE review_count > 10”. Therefore, we need index on attribute “review_count” in user table.

Add Index:

```
CREATE INDEX review_count_index ON user (review_count) USING BTREE;
```