ECE 356 Winter 2019: Lab2 Part2

Yelp

The general approach of how we add indexes and determine whether if those indexes are helpful in solving those problems:

1) Look at the output from the keyword “explain”, determine whether if the query involves table scan.
2) Look at the query, if it has the keywords: where, join on, order by, having, group by. The columns that involve those keywords are very likely to be the index.
3) After adding indexes, look at the output from the keyword “explain” again.
4) If the rows number reduced, the added indexes are useful for the problem solving.
5) If the rows number didn’t change, the added indexes are NOT useful for the problem solving.

Question 1

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

```
+----+-------------+-------+------+---------------+------+---------+------+---------+----------------+
| 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 (0.01 sec)
```

We can see there is a “Using filesort”. It is probably cause by the “order by”. So the review_count needs an index.

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

Question 2

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

```
+----+-------------+----------+------+---------------+------+---------+------+--------+----------------+
| 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)
```

We can see there is a “Using filesort”. It is probably cause by the “order by”. So the review_count needs an index.

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

Question 3

```
EXPLAIN SELECT AVG(individual_review_sum) AS users_average_review_count
FROM
  (SELECT Sum(review_count) AS individual_review_sum
   FROM user
   GROUP BY user_id) AS sum_list
```

```
+----+-------------+-------------+-------+---------------+---------+---------+------+---------+-------+
| id | select_type | table       | type  | possible_keys | key     | key_len | ref  | rows    | Extra |
+----+-------------+-------------+-------+---------------+---------+---------+------+---------+-------+
|  1 | PRIMARY     | <derived2>  | ALL   | NULL          | NULL    | NULL    | NULL | 1021667 | NULL  |
|  2 | DERIVED     | user        | index | PRIMARY       | PRIMARY | 22      | NULL | 1021667 | NULL  |
+----+-------------+-------------+-------+---------------+---------+---------+------+---------+-------+
2 rows in set (0.01 sec)
```

Again, derived table cannot be indexed and the second one is already indexed, so no indexing is needed for them. Form the query, since there is a “Sum(review_count)” and “AVG(individual_review_sum)” (Aggregation) the whole table will be scanned no matter what, so index will NOT help in this case.

Question 4

```
EXPLAIN SELECT count(*) AS users_count_difference_larger_than_point_five
FROM
  (SELECT user_id,
          avg(average_stars) AS average_star_user
   FROM user
   GROUP BY user_id) AS a_method
INNER JOIN
  (SELECT user_id,
          avg(stars) AS average_star_review
   FROM review
   GROUP BY user_id) AS b_method ON a_method.user_id = b_method.user_id
```

```
WHERE ABS(average_star_user - average_star_review) > 0.5;
```

```
+----+-------------+------------+-------+---------------+---------------+---------+-------------------+---------+-----------------------------------------------+
| id | select_type | table      | type  | possible_keys | key           | key_len | ref               | rows    | Extra                                         |
+----+-------------+------------+-------+---------------+---------------+---------+-------------------+---------+-----------------------------------------------+
|  1 | PRIMARY     | <derived2> | ALL   | NULL          | NULL          | NULL    | NULL              | 1021667 | NULL                                          |
|  1 | PRIMARY     | <derived3> | ref   | <auto_key0>   | <auto_key0>   | 22      | a_method.user_id  |      10 | Using where                                   |
|  3 | DERIVED     | review     | ALL   | NULL          | NULL          | NULL    | NULL              | 1655155 | Using temporary; Using filesort               |
|  2 | DERIVED     | user       | index | PRIMARY       | PRIMARY       | 22      | NULL              | 1021667 | NULL                                          |
+----+-------------+------------+-------+---------------+---------------+---------+-------------------+---------+-----------------------------------------------+
4 rows in set (0.00 sec)
```

Again, derived table cannot be indexed and the second one is already indexed, so no indexing is needed for them. There is a “Using filesort” on the third line. It is probably cause by the “order by”. So the review_count needs an index. However, form the query, since there is a “count(*)”, “avg(average_stars)” and “avg(stars)” (Aggregation), the whole table will be scanned no matter what, so even adding an index for review_count will NOT help in this case. So not index is needed.

Question 5

```
EXPLAIN SELECT
  (SELECT count(*)
   FROM
     (SELECT user_id,
             sum(review_count) AS user_review_sum_more_than_ten
      FROM user
      GROUP BY user_id
      HAVING user_review_sum_more_than_ten > 10) AS
user_review_sum_more_than_ten_table) /
  (SELECT count(*)
   FROM
     (SELECT user_id,
             sum(review_count) AS user_review_sum
      FROM user
      GROUP BY user_id) AS user_review_sum_table) AS fraction_of_user;
```

```
+----+-------------+------------+-------+---------------+---------+---------+------+---------+----------------+
| id | select_type | table      | type  | possible_keys | key     | key_len | ref  | rows    | Extra          |
+----+-------------+------------+-------+---------------+---------+---------+------+---------+----------------+
|  1 | PRIMARY     | NULL       | NULL  | NULL          | NULL    | NULL    | NULL |    NULL | No tables used |
|  4 | SUBQUERY    | <derived5> | ALL   | NULL          | NULL    | NULL    | NULL | 1021667 | NULL           |
|  5 | DERIVED     | user       | index | PRIMARY       | PRIMARY | 22      | NULL | 1021667 | NULL           |
|  2 | SUBQUERY    | <derived3> | ALL   | NULL          | NULL    | NULL    | NULL | 1021667 | NULL           |
|  3 | DERIVED     | user       | index | PRIMARY       | PRIMARY | 22      | NULL | 1021667 | NULL           |
+----+-------------+------------+-------+---------------+---------+---------+------+---------+----------------+
5 rows in set (0.01 sec)
```

Again, derived table cannot be indexed and the third and fifith one is already indexed and the first line is null, so no indexing is needed for them. Form the query, since there is a “count(*)” and “sum(review_count)” (Aggregation), the whole table will be

scanned no matter what, so even adding an index for review_count will NOT help in this case. So not index is needed.

Question 6

```
EXPLAIN SELECT avg(LENGTH) AS review_avg_length
FROM
  (SELECT char_length(text) AS LENGTH
   FROM
     (SELECT text
      FROM
        (SELECT user_id,
                sum(review_count) AS user_review_sum_more_than_ten
         FROM user
         GROUP BY user_id
         HAVING user_review_sum_more_than_ten > 10) AS
user_review_sum_more_than_ten_table
      INNER JOIN review ON review.user_id =
user_review_sum_more_than_ten_table.user_id) AS text_table) AS length_table;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	16551663	NULL
2	DERIVED	<derived3>	ALL	NULL	NULL	NULL	NULL	16551663	NULL
3	DERIVED	review	ALL	NULL	NULL	NULL	NULL	1655155	NULL
3	DERIVED	<derived4>	ref	<auto_key0>	<auto_key0>	22	Yelp.review.user_id	10	NULL
4	DERIVED	user	index	PRIMARY	PRIMARY	22	NULL	1021667	NULL

5 rows in set (0.01 sec)

Again, derived table cannot be indexed and the fifith one is already indexed, so no indexing is needed for them. For the third line, the review table needs an index. Form the query, it is also showing that the review.user_id after inner join need an index.

```
CREATE INDEX index_review_starReview ON review(user_id) USING BTREE;
CREATE INDEX index_review_starReview ON review(stars) USING BTREE;
```