Project 5

Due Apr 9, 2018 at 11:59pm **Points** 15 **Questions** 10 **Time Limit** None

Instructions

Please see [https://github.com/abadid/cmsc424-spring2018/tree/master/project5 (https://github.com/abadid/cmsc424-spring2018/tree/master/project5)](https://github.com/abadid/cmsc424-spring2018/tree/master/project5) for the complete project description.

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	218 minutes	11.5 out of 15

Score for this quiz: **11.5** out of 15
Submitted Apr 9, 2018 at 10:03pm
This attempt took 218 minutes.

Question 1 — 1 / 1 pts

Q1

If you notice the output, both queries, Q1.1 and Q1.2, return the same user. However, the first one finishes in much less time than the second. Why? If more than one answer is correct, choose the best of the correct answers (for this and all questions on this project):

- The id attribute appears before the username attribute in the table declaration, so it is faster to extract the id of a tuple than the username.
- Comparing strings is slower than comparing integers.
- This is arbitrary. If we change the constant value in the where clause, the results could be substantially different.

The first query ran earlier, so the table is still in the cache at the time that it ran.

Correct!

- [x] There is an index on id because it is declared as a PRIMARY KEY.

Question 2

1.5 / 1.5 pts

Q2:

Although the WHERE clause of both queries select the same number of tuples (100 tuples), and there is an index on both `id` and `username`, why does the first query finish in less time than the second one? (Choose the best answer)

The username index is on a string. Indexes on strings are much less useful than indexes on integers.

The WHERE clauses actually returns a different set of 100 tuples. It is the particular set of 100 tuples returned by Q2.2 that makes second query slow.

Correct!

- [x] Records are clustered according to id. Therefore the index on id is primary and the index on username is secondary. Range predicates are usually faster on primary indexes.

The table is sorted on id, therefore the index on id is unnecessary

Indexes on strings cannot be used for range predicates.

Question 3

1 / 1 pts

Q3: Part 1

Q3.1 and Q3.2 find one user in the table. However, the first one takes longer. Why? (Choose the best / most significant answer).

- 'Giant' is a shorter string than 'Richardson'. Therefore, the string comparison operations that happen when traversing the index are faster.
- The Postgres developers were misogynists, and biased the system to find males faster than females.

Correct!

- There are many more users having last name 'Richardson' compared to 'Giant', so the index is less helpful.
- The id of 'Patrick Giant' is much smaller than the id of 'Bethzy Richardson'. Hence PostgreSQL found it earlier.
- In the index, the record for 'Patrick Giant' appears much earlier than the record for 'Bethzy Richardson', and hence PostgreSQL found it earlier.

Question 4

1 / 1 pts

Q3: Part 2

Suppose instead of creating an index on `last_name`, we created it on `first_name`. Based on your understanding so far, what can you say about the run time of the two queries above, and why?

- Both queries will take the same time since the B-tree on last name will be balanced.

There will be no change, we will observe similar behavior as before

Correct!

The behaviour will be flipped, Q3.2 will take more time than Q3.1

Question 5

2 / 2 pts

Q4

Suppose instead of creating separate indexes on one of `first_name` or `last_name`, we create a _multicolumn_ index containing both attributes. Which of the following queries would be good candidates for using our new index (select all that apply)?

Correct!

SELECT username, first_name, last_name FROM users WHERE first_name = 'Bethzy' AND last_name = 'Smith'

SELECT username, first_name, last_name FROM users WHERE last_name = 'Giant'

Correct!

SELECT username, first_name, last_name FROM users WHERE first_name LIKE 'Jord%'

Correct!

SELECT username, first_name, last_name FROM users WHERE first_name = 'Jaxson'

SELECT username, first_name, last_name FROM users WHERE last_name LIKE 'Jord%'

Question 6

1 / 1 pts

Q5

Both queries, Q5.1 and Q5.2, return the same user record and can make use of at least one index. However, the second query takes longer to run than the first, why? (Choose the best answer):

- Usernames are larger (in terms of number of bytes) than states. Therefore the index on username is more helpful.
- The table is sorted by username; therefore selection predicates on username are faster.
- Q5.2 compares three column values for every user, whereas Q5.1 compares only one.

Correct!

- There are many user records that have state = 'CA', so the index on state is not particularly helpful.

Question 7

1 / 1 pts

Q6

Suppose we want to run many queries of the following type:
SELECT username, first_name, last_name
FROM users
WHERE date_of_birth >= '1990-01-01' AND date_of_birth <= '1990-02-01'
AND theme = 'B'

Assume that the range for `date_of_birth` is around a month (25-35 days).

Which one of the following index would be most helpful?

- CREATE INDEX users_dob_theme ON users (date_of_birth, theme);
- CREATE INDEX users_theme ON users (theme);
- Correct! CREATE INDEX users_theme_dob ON users (theme, date_of_birth);
- CREATE INDEX users_dob ON users (date_of_birth);

Question 8

1.5 / 1.5 pts

Q7

Q7.1 and Q7.2 both return the same value, but Q7.2 is slower than Q7.1. Why?

- PostgreSQL knows to stop when it finds one record in Q7.1, because there is a `UNIQUE` index on `username`. However, because the index on `about` is not unique, it reads more records than necessary.
- Although the value returned is the same, that is because this is an aggregate. The set of of records that are read from the users table are different. In particular, Q7.2 has to read more records.
- The 'username' attribute appears first in the record, before the 'about' attribute. Therefore, the cost of extracting the 'about' attribute is larger than the cost of extracting the 'username' attribute from each record in users.
- Correct! The cost to use the index on the about attribute is larger than the cost to use the index on the username attribute, since the index is larger and the search keys are larger.

Question 9

1.5 / 1.5 pts

Q8

Both queries, Q8.1 and Q8.2, update the id value of every single tuple in the table by concatening the year of birth with the old id. However, Q8.2 runs slower than Q8.1. Why?

- q8_users1 is smaller than q8_users2 since it doesn't have an index. Therefore q8_users1 blocks are more likely to be in cache or memory than q8_users2 blocks (which are more likey to be on disk).
- Because we didn't create an index on q8_users1, it automatically created an index on its primary key. This index helps to accelerate the update statements.

Correct!

- (selected) The update in Q8.2 is more expensive because the index has to be updated as well.
- Q8.2 runs after Q8.1 and the CPU on my computer slows down over time.

Question 10

0 / 3.5 pts

Q9

Please read the description of the problem in README carefully.

In this question, we created identical indexes on identical tables, with the only difference that uples are inserted in sorted order by id for users2 and (mostly) random order for users3.

Surprisingly, the index for which insertions happened in sorted order is **larger** than the the index for which insertions happened in **random** order. Please use your understanding of how insertions into indexes work from the

textbook and the lectures this semester to explain why there is a size difference between the two indexes.

Your Answer:

the structure of B tree will be organized differently when inserted in sorted order compare to insert in random order.

Quiz Score: **11.5** out of 15