SANJANA ADDAGARLA
TE IT
2018140002
BATCH A

Indexing

1. Use the script comments-ddl.sql provided to create a new table called comments.

   Creating a new table 'comments'.
   CREATE TABLE

   Query returned successfully in 163ms.
2. Use the script comments-insert.sql provided to populate large data (100000 rows) into the comments table (This may take about a minute or two to complete)

   INSERT 0 1

   Query returned successfully in 8 secs 979 msec.

3. Run the command analyze; to tell PostgreSQL to update statistics about tables; you should do this whenever you insert, delete or update a lot of data, otherwise the optimizer may use wrong statistics and choose bad query execution plans. Some databases such as SQL Server run an equivalent command automatically, but on others you have to run the analyze (or equivalent command) manually.

   ANALYZE

   Query returned successfully in 646 msec.

4. To record the time taken by a query, run it from pgAdmin3, and then select the Messages tab in the Output pane at the bottom. The execution time will be shown here. Now run the following queries and record the time taken; report the times in your submission. (Goal: to show a query whose plan uses an index, and another that cannot use any index and must do an expensive scan on the same relation, and show the difference in run times. Both queries retrieve at most a single row (by using a selection on primary key for the first query, and a selection on two columns, for the second query). Later we will see the actual query plans.)
   A. select * from comments where ID = 99999;
   B. select * from comments where rating = 5 and item_id = 99982;

TIME TAKEN

select * from comments where ID = 99999;	**select * from comments where rating = 5 and item_id = 99982;**
Successfully run. Total query runtime: 76 msec. 1 rows affected.	Successfully run. Total query runtime: 116 msec. 1 rows affected.
Successfully run. Total query runtime: 73 msec. 1 rows affected.	Successfully run. Total query runtime: 109 msec. 1 rows affected.
Successfully run. Total query runtime: 136 msec. 1 rows affected.	Successfully run. Total query runtime: 85 msec. 1 rows affected.
Successfully run. Total query runtime: 72 msec. 1 rows affected.	Successfully run. Total query runtime: 87 msec. 1 rows affected.
Successfully run. Total query runtime: 73 msec. 1 rows affected.	Successfully run. Total query runtime: 90 msec. 1 rows affected.

We see from the above table of comparison that the first query takes overall less total query time as compared to the second query. This is because the first query plan uses primary key as an index while the second query uses non-primary key attributes, that is does not use an index and hence has to perform more extensive searches to achieve a result which leads to greater total query times.

5. Find the query plan for each of the above queries, by prefixing the query with the explain keyword.
   A. explain select * from comments where ID = 99999;

   c1=0.29
   c2=8.31

   Index Scan using comments_pkey on comments (cost=0.29..8.31 rows=1 width=78)
     Index Cond: (id = 99999)
   Successfully run. Total query runtime: 66 msec.
   2 rows affected.

B. explain select * from comments where rating = 5 and item_id = 99982;

c1=0.00
c2=2857.00

Seq Scan on comments (cost=0.00..2857.00 rows=1 width=78)
  Filter: ((rating = 5) AND (item_id = 99982))
Successfully run. Total query runtime: 71 msec.
2 rows affected

Now create an index on the unindexed attribute rating of the comments relation by executing create index comments_rating on comments(rating);
Rerun the preceding queries from step 4 and record the time as well as the query plan.

create index comments_rating on comments(rating);
CREATE INDEX

Query returned successfully in 208 msec.

select * from comments where ID = 99999;	**select * from comments where rating = 5 and item_id = 99982;**
Successfully run. Total query runtime: 93 msec. 1 rows affected.	Successfully run. Total query runtime: 79 msec. 1 rows affected.

A. explain select * from comments where ID = 99999;

c1=0.29
c2=8.31

Index Scan using comments_pkey on comments (cost=0.29..8.31 rows=1 width=78)
  Index Cond: (id = 99999)
Successfully run. Total query runtime: 114 msec.
2 rows affected.

B. explain select * from comments where rating = 5 and item_id = 99982;

c1=366.52
c2=2015.72

Bitmap Heap Scan on comments (cost=366.52..2015.72 rows=1 width=78)
  Recheck Cond: (rating = 5)
  Filter: (item_id = 99982)
  -> Bitmap Index Scan on comments_rating (cost=0.00..366.52 rows=19480 width=0)
        Index Cond: (rating = 5)
Successfully run. Total query runtime: 86 msec.
5 rows affected.

Now similarly create an index on item_id attribute of the comments relation.
create index comments_item_id on comments(item_id);

CREATE INDEX

Query returned successfully in 138 msec.

ANALYZE

Query returned successfully in 356 msec.

select * from comments where ID = 99999;	**select * from comments where rating = 5 and item_id = 99982;**
Successfully run. Total query runtime: 75 msec. 1 rows affected.	Successfully run. Total query runtime: 77 msec. 1 rows affected.
Successfully run. Total query runtime: 80 msec. 1 rows affected.	Successfully run. Total query runtime: 82 msec. 1 rows affected.
Successfully run. Total query runtime: 74 msec. 1 rows affected.	Successfully run. Total query runtime: 79 msec. 1 rows affected.

We see that the query time for the second query has significantly gone down as compared to the previous case, this is due to the face that now as we have created a new index on item_id, the second query too like query one searches based on the index and hence query time is reduced.

A. explain select * from comments where ID = 99999;

c1=0.29
c2=8.31

Index Scan using comments_pkey on comments (cost=0.29..8.31 rows=1 width=78)
  Index Cond: (id = 99999)
Successfully run. Total query runtime: 90 msec.
2 rows affected.

B. explain select * from comments where rating = 5 and item_id = 99982;

c1=0.29
c2=8.31

Index Scan using comments_item_id on comments (cost=0.29..8.31 rows=1 width=78)
  Index Cond: (item_id = 99982)
  Filter: (rating = 5)
Successfully run. Total query runtime: 70 msec.
3 rows affected.

Find the plans for the following queries (First one uses a condition only on an indexed attribute, second one uses one conjunct on an indexed attribute and one conjunct on an unindexed one)

- **explain select * from comments where ID = 99999;**
  c1 = 0.29, c2 = 8.31
  Index Scan using comments_item_id on comments (cost=0.29..8.31 rows=1 width=78)
  Index Cond:(id=99999)

  Successfully run. Total query runtime: 70 msec.
  2 rows affected.

- **explain select * from comments where rating = 5 and item_id = 99982;**
  (c1=0.29, c2=8.31)
  Index Scan using comments_item_id on comments (cost=0.29..8.31 rows=1 width=78)

Index Cond:(id=99982)
Filter:(rating = 5)

Successfully run. Total query runtime: 66 msec.
3 rows affected.

- **explain select * from comments where rating = 5;**

  Bitmap Heap Scan on comments (cost=377.87..1982.37 rows=19800 width=78)
  Recheck Cond: (rating = 5)
  -> Bitmap Index Scan on comments_rating (cost=0.00..372.92 rows=19800 width=0)
  Index Cond: (rating = 5)

- **explain select * from comments where rating = 5 and comment = 'Q';**

  Bitmap Heap Scan on comments (cost=372.93..2026.93 rows=47 width=78)
  Recheck Cond: (rating = 5)
  Filter: (comment = 'Q'::text)
  -> Bitmap Index Scan on comments_rating (cost=0.00..372.92 rows=19800 width=0)
  Index Cond: (rating = 5)

  Successfully run. Total query runtime: 79 msec.
  5 rows affected.

CONCLUSION:

We see that time required to create index on item_id(138ms) is less than that required by rating(208ms).

We also observe that cost and query time for the queries is less in the case where item_id is used as index as when ratings is used as an index.