Database Systems, CSCI 4380-01
Homework # 7
Due Monday November 29, 2021 at 11:59:59 PM

Homework Statement. This homework is worth 5% of your total grade. This homework is completely optional. If you skip this homework, the final exam will be worth 5 points more. However, I highly recommend that you solve part or all of this homework.

This homework will concentrate on query processing and cost estimation.

For this homework, I created the full radiodb database as `radiodb_full` in the db server. The schema is the same as previous homeworks, just a much larger database.

Since this homework involves calculations of cost based on how the database is stored, we need to all use the same version of the database, so no local databases.

There is already a B-tree for all the primary keys. Additionally, I have created the following indices:

```
create index pidx1 on playedonradio(station, playedtime);
create index pidx2 on playedonradio(songid);
create index pidx3 on playedonradio(station, songid, playedtime);
create index sidx1 on spotify(songid);
create index sidx2 on spotify(position, streamdate);
create index sidx3 on spotify(streamdate, songid, position);
```

Problem Description

For each query below, estimate its cost by using index scan over one applicable index over the `radiodb_full` database. An index applicable if it contains at least one attribute that is used in a condition in the WHERE statement of the query. Repeat this for all possible indices.

For example, given the query: `select song_id from playedonradio where station='mai';`, you are looking for any index that has the attribute `playedonradio(station)` anywhere in the indexed attributes (in case `pidx1` and `pidx3`).

Each cost estimate must use only one index, scan the index and then the relation if needed. Find how many pages need to be scanned from the index or the relation by querying the database to find the number of matching tuples for a query (using `select count(*)`). Note that the data is very large, so avoid large queries as much as possible.

You must estimate the cost by making the following assumptions:

1. All costs are in terms of disk pages, cost of scanning the index and cost of reading matching tuples from the relation if needed.

2. If reading tuples from a relation, use the worst case scenario: all tuples are in a different disk page.

   If the number of expected tuples in the output is higher than the number of pages, you can still use the number of tuples in your cost computation since some pages may need to be read more than once. In such a case, obviously index scan should not be used and the query optimizer will be able to decide on this based on the large cost.

3. Assume all indices have 3 levels (root, internal node and leaf level). Assume the root of all indices are in memory so scanning the root incurs no cost. Most searches will have 1 node from the internal level and a number of leaf nodes.

4. Postgresql returns the size of an index in terms of number of pages using the query below. For the B-tree index, assume the numbers provided are the number of leaf nodes.

```
SELECT relname, relpages
FROM pg_class pc, pg_user pu
WHERE pc.relowner = pu.usesysid and pu.usename = 'dbclass'
ORDER BY relname desc;
```

Queries

```
-- Q1
select songid from playedonradio
where station = 'mai' ;

-- Q2
select songid from playedonradio
where playedtime >= timestamp '2020-11-18 00:00:00'
     and playedtime <= timestamp '2020-11-18 23:59:59';

-- Q3
select songid from playedonradio
where playedtime >= timestamp '2020-11-18 08:00:00'
     and playedtime <= timestamp '2020-11-18 09:59:59'
     and station = 'mai';

-- Q4
select playedtime from playedonradio
where songid = 29643;

-- Q5
select songid playedonradio
where songid = 29643
     and station = 'george';

-- Q6
select id playedonradio
where songid = 29643
     and station = 'george'
     and playedtime::date = date '2020-11-14';

-- Q7
select streams from spotify
where songid = 13154;
```

```
-- Q8
select songid from spotify
where streamdate = date '11-01-2020' ;

-- Q9
select streamdate from spotify
where position = 1
      and songid = 13154;

-- Q10
select streamdate from spotify
where position = 1
      and songid = 13154
      and streamdate >= date '2020-01-01';
```

SUBMISSION INSTRUCTIONS. You will use SUBMITTY for this homework. Submit a single text file named `hw7.txt` that lists the cost of each query for a single index on a single line with the following format (no spaces):

`query id,index name,index_pages_scanned,relation_pages_scanned`

For example, if you are given the following queries:

```
qa: select songid from playedonradio where id >= 1000000 and id <= 1000010;
qb: select playedtime from playedonradio where songid=67546;
```

Your output would contain at least the following lines:

```
qa,playedonradio_pkey,2,11
qb,pidx2,3293,0
```

`qa` estimate is based on the 11 tuples matching this query, which takes 1 internal node, 1 leaf node to scan the index, and 11 relation pages in the worst case.

`qb` estimate corresponds to 8 tuples matching this query (1 internal node, and all leaf nodes) and the index contains all the necessary information (so zero relation tuples need to be read).

Some final thoughts

Now that you have seen what these queries cost and how costs compare given the various indices, do you think the database will use a specific index? Is it better than sequential index? Which index is the best?

You can quickly test this out, by looking at query plans. Simply add the word `explain` before each query to look at the query plan. For example, for the above query `qb`, the query plan does not use the index at all. Why is this the case?

```
radiodb_full=> explain select playedtime from playedonradio where songid=67546;
                        QUERY PLAN
----------------------------------------------------------------
 Bitmap Heap Scan on playedonradio (cost=5.18..361.72 rows=98 width=8)
   Recheck Cond: (songid = 67546)
   ->  Bitmap Index Scan on pidx2 (cost=0.00..5.16 rows=98 width=0)
         Index Cond: (songid = 67546)
(4 rows)
```

In addition to the given queries, try them in queries joining with other relations.