Database Systems, CSCI 4380-01
Homework # 7
Due Thursday December 3, 2020 at 11:59:59 PM

Homework Statement. This homework is worth 6% of your total grade. If you choose to skip it, Final Exam will be worth 6% more.

In this homework, you are asked to estimate the cost of queries over the full streaming database. Connect to the class server at: `http://rpidbclass.info` for the database `streaming`.

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 series_idx1 on series(rottentomatoes) ;
create index series_idx2 on series(imdbrating) ;
create index series_idx3 on series(contentrating, title) ;
create index series_idx4 on series(yearreleased) ;
create index series_idx5 on series(contentrating, rottentomatoes, imdbrating, title) ;
create index seriescast_idx1 on seriescast(castname) ;
create index seriescast_idx2 on seriescast(castname, seriesid) ;
create index seriescountry_idx1 on seriescountry(country) ;
create index seriescountry_idx2 on seriescountry(country, seriesid) ;
```

Problem Description

For each query below, estimate its cost by using one applicable index. An index applicable if it contains an attribute that is used in a condition in the query. Repeat this for all possible indices.

For example, given the query: `select title from series where imdbrating=8;`, you are looking for any index that has the attribute `imdbrating` anywhere in the indexed attributes. For example, for this query, you can use `series_idx2` or `series_idx5`. So, estimate the cost using only `series_idx2` first, then do another cost estimate using only `series_idx5`.

For the query: `select seriesid from seriescountry where country = 'USA'` you can use `seriescountry_idx1`, `seriescountry_idx2` or `seriescountry_pkey` (the primary key index).

For many queries, there will be two applicable indices, but a few will have more. Also, at times you can answer queries using index only scan.

Each cost estimate must use only one index, scan the index and then the relation if needed. Find how many tuples need to be scanned from the index or the relation by querying the database (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.

3. Assume all indices have 3 levels (root, internal node and leaf level), and 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 relpages desc;
```

Queries for the homework:

```
-- q1
select title from series where yearreleased = 2020;

-- q2
select title from series where yearreleased <= 1950 and yearreleased >= 1941;

-- q3
select title from series where rottentomatoes <= 100 and rottentomatoes >= 95;

-- q4
select title from series where rottentomatoes <= 60 and rottentomatoes >= 55;

-- q5
select title from series where imdbrating <= 6 and imdbrating >= 5.5;

-- q6
select title from series where imdbrating <= 100 and imdbrating >= 9.5;

--q7
select title from series where contentrating = 'all' and imdbrating >= 9;

--q8
select title from series where
        rottentomatoes <= 60 and rottentomatoes >= 55
        and imdbrating<= 6 and imdbrating>= 5.5
        and contentrating = 'all';

--q9
select seriesid from seriescast where castname = 'David Attenborough';

--q10
select seriesid from seriescast where castname = 'Jodie Whittaker';

--q11
select seriesid from seriescountry where country = 'Japan';

--q12
select seriesid from seriescountry where country = 'Greece';
```

SUBMISSION INSTRUCTIONS. 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 the following query `qa` was input:

```
select director from seriesdirectors where seriesid<= 100;
```

then the output in your submission for this query would be:

`qa,seriesdirectors_pkey,3,0`

Because there are 5 tuples matching this query (1 internal node, and 2 leaf nodes in the worst case) and the index contains all the necessary information (so zero 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?

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 `aq`, the query plan does not use the index at all. Why is this the case?

```
streaming=> explain select director from seriesdirectors where seriesid<= 100;
                         QUERY PLAN
------------------------------------------------------------
 Seq Scan on seriesdirectors (cost=0.00..2.62 rows=4 width=13)
   Filter: (seriesid <= 100)
(2 rows)
```

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

```
select s.title from series s, seriescast sc
where s.seriesid = sc.seriesid and sc.castname = 'David Attenborough';

select s.title from series s, seriescast sc
where s.seriesid = sc.seriesid and sc.castname = 'Jodie Whittaker';

select s.title from series s, seriescountry sc
where s.seriesid = sc.seriesid and sc.country = 'Japan';

select s.title from series s, seriescountry sc
where s.seriesid = sc.seriesid and sc.country = 'Greece';
```

This is a great way to learn about optimization, by doing some sanity check.

Database Schema

```
-- TV series on various platforms
create table series (
      seriesid int  primary key
      , title varchar(400)
      ,  yearreleased  int
      ,  contentrating varchar(40) -- age group the movie is intended for
      ,  imdbrating    float       -- imdb rating
      ,  rottentomatoes int        -- rotten tomatoes rating
      ,  description   text
      ,  seasons       int         -- how many seasons are available
      ,  date_added    date        -- date series is added to Netflix
) ;

-- People who starred in the series
create table seriescast (
      seriesid int
      , castname varchar(100)
      , primary key (seriesid, castname)
      , foreign key (seriesid) references series(id)
) ;

-- Countries the series is available in
create table seriescountry (
      seriesid int
      , country varchar(100)
      , primary key (seriesid, country)
      , foreign key (seriesid) references series(id)
) ;
```