PS 2 - Query Optimization

Database Systems, CS-UH 2214

Assigned:	April 6th, 2022	Due:	April 20th, 2022

Some questions in this problem set are modified from Sam Madden's Principles of Database Systems course taught @ MIT.

1 Query Plans

In this problem set, you will work with a publicly available dataset on health records - MIMIC-2. The goal of the problem set is to get hands on experience learning the different optimizations made by a database. Download the mimic2 file in the source files folder and unzip it. After that, create a database called 'mimic2' on your psql set up. Navigate to the folder of the unzipped file and type in the following command.

```
psql -U postgres mimic2 < mimic2
```

This process may take a lengthy amount of time, but it will create a PostgreSQL database on your local server. Once the database is created, you may need to set your search path to the correct schema. To do this, run:

```
SET search_path to mimic2v26;
```

This process is straightforward for macOS/Unix systems, but fairly complicated for Windows users (Physionet do not provide scripts for Windows systems).

1.1 Query Plans

For the following two queries, run them (using EXPLAIN) in PostgreSQL and answer questions (a) - (f) below:

```sql
SELECT DISTINCT ce2.subject_id, ce2.value1
FROM chartevents AS ce1, d_chartitems AS ci1,
     chartevents AS ce2, d_chartitems AS ci2
WHERE ce1.itemid=ci1.itemid
    AND ce2.itemid=ci2.itemid
    AND ce1.subject_id=ce2.subject_id
    AND ci1.label='Diagnosis/op'
    AND ce1.value1='CONGESTIVE HEART FAILURE'
    AND ci2.label='Diagnosis/op'
    AND ce2.value1!='CONGESTIVE HEART FAILURE';
```

```sql
SELECT ci.label, COUNT(*) AS freq
FROM d_chartitems AS ci, chartevents AS ce
WHERE ci.itemid=ce.itemid
GROUP BY ci.label
ORDER BY freq DESC
LIMIT 20;
```

a. What physical plan does PostgreSQL use? Your answer should consist of a drawing of a query tree annotated with the access method types and join algorithms. **[1 point]**

b. Why do you think PostgreSQL selected this particular plan? **[1 point]**

c. What does PostgreSQL estimate the size of the result set to be? **[1 point]**

d. When you actually run the query, how big is the result set? **[1 point]**

e. Run some queries to compute the sizes of the intermediate results in the query. Where do PostgreSQL's estimates differ from the actual intermediate result cardinalities? **[1 point]**

f. Given the tables and indices we have created, do you think PostgreSQL selected the best plan? You can list all indices with \di, or list the indices for a particular table with \d tablename. If not, what would be a better plan? Why? **[1 point]**

2 Estimating the cost of query plans

Use EXPLAIN to find the query plans for the following two queries and answer question (a).

```sql
SELECT l.test_name
FROM d_chartitems AS c, chartevents AS ce,
     labevents AS le, d_labitems AS l
WHERE c.itemid=ce.itemid
    AND ce.subject_id=le.subject_id
    AND le.itemid=l.itemid
    AND ce.charttime > '3000-02-17 00:00:00'
    AND l.itemid>50700;
```

```sql
SELECT l.test_name
FROM d_chartitems AS c, chartevents AS ce,
     labevents AS le, d_labitems AS l
WHERE c.itemid=ce.itemid
    AND ce.subject_id=le.subject_id
    AND le.itemid=l.itemid
    AND ce.charttime > '3600-02-17 00:00:00'
    AND l.itemid>50700;
```

a. Notice that the query plans for the two queries above are different, even though they have the same general form. Explain the difference in the query plan that PostgreSQL chooses, and explain why you think the plans are different. **[3 points]**

Now use EXPLAIN to find the query plan for the following query, and answer questions (b) - (g).

```sql
SELECT l.test_name
FROM d_chartitems AS c, chartevents AS ce,
     labevents AS le, d_labitems AS l
WHERE c.itemid=ce.itemid
    AND ce.subject_id=le.subject_id
    AND le.itemid=l.itemid
    AND ce.charttime > '3500-02-17 00:00:00'
    AND l.itemid>50700;
```

b. What is PostgreSQL doing for this query? How is it different from the previous two plans that are generated? You may find it useful to draw out the plans (or use pgadmin3) to get a visual representation of the differences, though you are not required to submit drawings of the plans in your answer. **[2 points]**

c. Run some more EXPLAIN commands using this query, sweeping the constant after the '>' sign from 3000 to 3600. For what value of charttime does PostgreSQL switch plans? Note: There might be additional plans other than these three. If you find this to be the case, please write the estimated last value of charttime for which PostgreSQL switches query plans. **[2 points]**

d. Why do you think it decides to switch plans at the values you found? Please be as quantitative as possible in your explanation. **[2 points]**

e. Suppose the crossover point (from the previous question) between queries 2 and 3 is charttime23. Compare the actual running times corresponding to the two alternative query plans at the crossover point. How much do they differ? Do the same for all switch points. Inside psql, you can measure the query time by using the \timing command to turn timing on for all queries. To get accurate timing, you may also wish to redirect output to /dev/null, using the command \o /dev/null; later on you can stop redirection just by issuing the \o command without any file name. You may also wish to run each query several times, throwing out the first time (which may be longer if the data is not resident in the buffer pool) and averaging the remaining runs. **[2 points]**

f. Based on your answers to the previous two problems, are those switch points actually the best place to switch plans, or are they overestimate/underestimate of the best crossover points? If they are not the actual crossover point, can you estimate the actual best crossover points without running queries against the database? State assumptions underlying your answer, if any. **[2 points]**

3 Access Methods

Your database system supports heap files and B+-trees (clustered and unclustered). B+-tree leaf pages point to records in the heap file. Assume that you can have at most one clustered index per file, and that the database system has up-to-date statistics on the cardinality of the tables, and can accurately estimate the selectivity of every predicate. Assume B+-tree pages are 50% full, on average. Assume disk seeks take 10 ms, and the disk can sequentially read 100 MB/sec. In your calculations, you can assume that I/O time dominates CPU time (i.e., you do not need to account for CPU time.)

Disk pages are 4 KB in size. For each query, assume that no pages are already in memory. Assume that during the query, any page that has been read remains in the buffer pool until the end of the query (no pages are evicted).

You will estimate the performance of this database system when running queries over a crowdfunding database similar to Kickstarter.

The database contains the following tables:

```
--- Each funder has a unique id did and a name
funders (fid int PRIMARY KEY,
        name char(100))

--- Each project has a unique id pid, a project desc and a budget goal
projects (pid int PRIMARY KEY,
          desc char(500),
          budget int)

--- A funder visits a project page
visits (vid int PRIMARY KEY,
           v_pid int REFERENCES projects(pid),
           v_fid int REFERENCES funders(fid),
           v_time timestamp)

--- A funder can comment, like, share or contribute to a project during a visit
events(eid int PRIMARY KEY,
        e_vid int REFERENCES visits(vid),
        type char(20),
        e_time timestamp)

--- contributions to a project
contributions(cid int PRIMARY KEY,
      c_eid int REFERENCES events(eid),
      amt int)
```

In this database, int and timestamp values are 8 bytes each and characters are 1 byte. All tuples have an additional 8 byte header. This means, that, for example, the size of a single funders record is 8 + 8 + 100 = 116 bytes.

For the queries below, you are given the following statistics:

In the absence of other information, assume that attribute values are uniformly distributed.

a. Suppose the system is running the query

Statistic	**Value**
# of funders	10^5
# of projects	10^3
# of visits	10^6
# of events	10^7
# of contributions	10^5
Time span of events in the database	Jan 1, 2018 - Jan 1, 2020

```
SELECT * FROM events where eid = 548291132.
```

Suppose there is a B+Tree on events.eid. Considering all possible plans the executor might use, estimate the runtime of the fastest plan to evaluate this query when the index is unclustered. **[1 point]**

b. Consider the same query and the same B+Tree on events.eid. Considering all possible plans the executor might use, estimate the runtime of the fastest plan to evaluate this query when the index is clustered. **[1 point]**

For the following questions, assume the database system is running the following query that finds the names of funders with the top 10 most eventful visits, in terms of number of contributions during 2015.

```
SELECT name, count FROM funders,
  (SELECT vid, v_fid, COUNT(*) AS count
   FROM visits, events, contributions
   WHERE vid = e_vid
   AND c_eid = eid
   AND v_time BETWEEN '1/1/2019' and '31/12/2019'
   GROUP BY vid, v_fid
   ORDER BY COUNT(*) DESC
   LIMIT 10)
AS max
WHERE fid = v_fid;
```

c. Suppose only heap files are available (i.e., there are no indexes), and that the system has both grace (hybrid) hash and merge join methods available to it. Draw the query plan you think the database system would use to evaluate this query. **[1 point]**

d. For each node in your query plan indicate (on the drawing, if you wish), the approximate output cardinality (number of tuples produced.) **[1 point]**

e. Estimate the runtime of the plan you drew, in seconds. **[1 point]**

f. Now, suppose that there are clustered B+Trees on fid, vid, eid and cid, and an unclustered B+Tree on e_vid. Draw the new plan you think the database would use and estimate its runtime. **[1 point]**

g. Suppose that there are clustered B+Trees on fid, v_time, e_vid and c_eid, and unclustered B+Trees on cid and vid. Draw the plan you think the database would use and estimate its runtime. **[1 point]**

h. Is it possible to flatten the query in the previous problem into an un-nested representation? If so, write the flattened representation. If not, state why. **[1 point]**

3.1 Submission

You will package your entire solution into a single PDF file with the title: 'PS2-netID1-netID2-netID3'. Submit the solution file via Dropbox using this link.

`https://bit.ly/CSUH2214-S22-PS2`

Only one student per group should submit and that student should always submit/resubmit to avoid multiple submissions per group.

In your PDF file, please include a comma separated list of each member's full name.

We will not grade any submission that does not strictly follow the submission rules.

3.2 Collaboration

This lab should be manageable for a single person, but if you prefer to work in a group of up to **three** students, this is also OK. Larger groups are not allowed.

If you choose to work in a group, you need to determine how best to meet and work together ***online****. In line with the university policy, in-person gatherings are discouraged. If you cannot agree on a suitable and safe online meeting method before starting the problem set, we strongly discourage you from working in a group.*