Homework 4: Query Processing, Query Optimization
(due March 21st, 2016, 4:00pm, in class—hard-copy please)

Reminders:

a. Out of 100 points. Contains 6 pages.
b. Rough time-estimates: 6~8 hours.
c. Please type your answers. Illegible handwriting may get no points, at the discretion of the grader. Only drawings may be hand-drawn, as long as they are neat and legible.
d. There could be more than one correct answer. We shall accept them all.
e. Whenever you are making an assumption, please state it clearly.
f. Lead TA for this HW: Sorour Amiri.
g. If you have any problems with the server, contact TAs (Sorour Amiri/Shamimul Hasan).

Q1. Relational Operators [25 points]

Consider the natural join $R \bowtie S$ of relations R(a, b, c) and S(c, d, e) given the following:

- Relation R contains NR = 60,000 tuples with 15 tuples per page.
- Relation S has NS = 5,000 tuples with 100 tuples per page.
- B = 201 buffer pages are available.

Assume that both relations are stored as simple heap files and that neither relation has any indexes built on it.

Q1.1. (5 points) Which is the smaller relation i.e. which is the one with the fewer number of pages? Write down the number of pages in the smaller relation (call it M) and the number of pages in the larger relation (call it N).

Q1.2. (20=5x4 points) Find the costs (in terms of disk accesses) of the following joins (make sure you also write the formulae you use for each one in terms of M, N, and B):

A. Nested Loops Join (page 454 of textbook or check slides)

B. Block nested loops Join (page 455 of textbook or check slides)

C. Sort-Merge Join (page 461 of textbook or check slides)

D. Hash Join (page 464 of textbook or check slides)

Q2. Query Optimization (Shoe Store) [25 points]

Consider the following schema of a shoe store database:

Store(sid, location)

ShoeType(id, style, size, color)
Inventory(type, sid, quantity)

Table "Store" has the data of each shoe store in the database which contains a unique id and the location. The "ShoeType" table has the data of each possible type of shoe in the database. This table keeps a unique value as the id, the style, size and the color of the shoe. The "Inventory" table listed the number of shoe types in each store.

"Inventory.type" is a foreign key to "ShoeType.id" and "Inventory.sid" is a foreign key to "Store.sid". We are given the following information about the database: Store contains 500 records with 10 records per page. ShoeType contains 1000 records with 40 records per page. Inventory contains 50,000 records with 50 records per page.

There are 100 values for Inventory.type.
There are 50 value for Store.location
There are 10 values for ShoeType.size (5,6,…,14)
There are 1000 values for Inventory.quantity

Consider the following queries:

Query 1:
```
SELECT sid
FROM Store where location='Blacksburg';
```

Query 2:
```
SELECT S.sid, S.location, T.size
FROM Store S, ShoeType T, Inventory I
WHERE S.sid=I.sid AND I.type = T.id;
```

Query 3:
```
SELECT S.sid, S.location, T.size
FROM Store S, ShoeType T, Inventory I
WHERE S.sid=I.sid AND I.type = T.id AND T.size>7 AND I.quantity = 20;
```

Q2.1. (5 points) Assuming uniform distribution of values, estimate size of the result for Query 1 (number of tuples).

Q2.2. (5 points) Again assuming uniform distribution of values and column independence, estimate the number of tuples returned by *Query 3*. Also assume you are given that the size of *Query 2* is 50,000 tuples. (*Hint*: Estimate the selectivity of T.size>7 AND I.quantity = 20)

Q2.3. (5 points) Draw all possible unique left-deep join query trees for Query 2.

Q2.4. (10 points) From your answer in Q2.3, consider only those trees where the smallest relation is the left most relation. Now, for the *first* join in each such tree

(i.e. the join at the bottom of the tree), what join algorithm would work best (i.e. cost the least)? Assume that you have 70 pages of buffer memory. There are no indexes; so indexed nested loop is not an option. Consider the Page Oriented Nested Join (PNJ), Block-NJ, Sort-Merge-J, and Hash-J. Make sure you show your work, and also write down the formula you use for each case (check slides).

Q3. Query Optimization (Stats and Range Queries) [20 points]

Consider the following three relations:

- cdtoc(id, release, discid, track_count, leadout_offset, track_offset)
- release(id, title, artist, year, comment_count, catalogue, source, barcode)
- track(id, release, title, artist, sequence)

The above information was gathered from the MusicBrainz Database where various pieces of information about music are stored, from artists and their releases to works and their composers. The "cdtoc" table is the table of contents of each CD. In the "release" table we keep the information of any released album. The "track" table has the information of tracks inside the released albums. Note that we have slightly modified the dataset to make it more understandable.

Assume there are no existing indexes on these tables and relations are stored as simple heap files.

These three tables are stored in cs4604.cs.vt.edu server. This is the first time you will be accessing the PostgreSQL server, so refer to the guidelines here (account information etc.):

http://courses.cs.vt.edu/~cs4604/Spring16/project/postgresql.html

Use the following commands to copy the tables to your private database. Note that you need to run these commands at the *command prompt* of cs4604.cs.vt.edu, NOT at the psql prompt (i.e. run these on the first prompt you get after ssh-ing to the server):

- pg_dump -U YOUR-PID -t cdtoc cs4604s16 | psql -d YOUR-PID
- pg_dump -U YOUR-PID -t release cs4604s16 | psql -d YOUR- PID
- pg_dump -U YOUR-PID -t track cs4604s16 | psql -d YOUR-PID

Sanity Check: run the following two statements and verify the output.

- select count(*) from cdtoc; // output – count = 284301
- select count(*) from release; // output – count = 284301
- select count(*) from track; // output – count = 3409654

For this question, it may help to familiarize you with the pg_class and pg_stats tables, provided by PostgreSQL as part of their catalog. Please see the links below:

http://www.postgresql.org/docs/8.4/static/view-pg-stats.html

http://www.postgresql.org/docs/8.4/static/catalog-pg-class.html

Now answer the following questions:

Q3.1. (6 points) Using a single SQL SELECT query on the 'pg_class' table, for each relation 'cdtoc', 'release' and 'track' to find (a) the number of rows, (b) the number of attributes and (c) if the relation has a primary key. Write down the SQL query you used and also paste the output.

Q3.2. (8 points) We want to find the top five years in which most albums are added to the release table. There are two ways to do that:

A. (4 points) Write a SQL query that uses the release table find the top 5 years in which most albums are added to the release table. Paste the output too when you run the query.

B. (4 points) Now write a SQL query that uses only the 'pg_stats' table instead to find the top 5 years in which albums are added to the release table. Again, paste the output.

Hint 1: You may find the 'most_common_vals' column in the table 'pg_stats' useful.

Hint 2: to get the first three elements of a 'list' type (e.g. 'mylist'), you can use 'mylist[1:5]', and to get the first three elements of 'anyarray' type (e.g. 'myarry'), you can use (myarray::varchar::varchar[])[1:5] see this link: http://www.postgresql.org/docs/8.4/static/view-pg-stats.html

Q3.3. (6 points) Consider the following query that retrieves all the albums which are added after 2010. *Note:* You do not need to run anything on the server for this part.

Query 3:
```
SELECT *
FROM release
WHERE year > 2010;
```

A. (3 points) Recall there is no index on year attribute. Will the optimizer use an Index scan or a simple File Sequential Scan? Explain in only 1 line.

B. (3 points) What would happen if someone creates a non-clustered B+-Tree index on 'year' attribute? Will it help to speed up the retrieval? Again explain in only 1.

Q4. Query Optimization (Joins) [30 points]

Again consider the same three relations and the database tables given in Q3 before. In this question, we want to explore the effect of creating indexes on join optimization. Again start with the assumption that there are no indexes on the tables.

It will help to familiarize yourself with the EXPLAIN and ANALYZE commands of PostgreSQL. Please visit the links given at the end for the same and study how to run it on a given SQL query and what output these commands return.

Q4.1. (5 points) Consider the same Query 3 from Q3.3 in the previous page. Run the 'explain analyze' command sequence on it and answer the following questions.

A. (1 point) Copy-paste the output of using EXPLAIN ANALYZE on Query 3.

B. (2 points) What is the estimated result cardinality of Query 3?

C. (2 points) What is the number of rows Query 3 actually returns?

Note: No need to paste the actual output rows of Query 3 since it will be too large. We just want the output of running EXPLAIN ANALYZE.

Q4.2 (10 points) Consider the following query

Query 4:

```
SELECT release.title, track.artist
FROM release,track , cdtoc
WHERE release.title = track.title
        AND release.year >= 2011
        AND release.catalogue = 5
        AND release.comment_count = 10;
```

The catalogue is a number that can often be found on the spine or near the barcode. The comment_count records the number of comments that are on a released album in the database.

Run the 'explain analyze' command sequence on Query 4 and get the query plan returned by the optimizer to answer the following questions.

A. (2 points) Explain what does Query 4 do?

B. (1 points) Copy-paste the output of using EXPLAIN ANALYZE on Query 4.

C. (5 points) Draw the query plan returned using tree and relational algebra notations as given in lecture slides.

D. (2 points) Report the estimated result cardinality, actual result cardinality (i.e. the true number of rows returned) and the total runtime of executing Query 4.

Note: Again, we do not want the actual output rows of Query 4 itself since it will be too large.

Q4.3 (5 points) For the Query 4, which attributes for the three tables you think should have index? Create index on these three tables for the proper attributes. Write the SQL queries you used to create these indexes (give these indexes any names you want).

Q4.4 (10 points) Update the statistics using the VACCUM and ANALYZE commands. Now run the EXPLAIN ANALYZE command again on the Query 4 given in Q4.2 and get the new query plan returned by the optimizer to answer the following questions.

A. (2 points) Copy-paste the output of using EXPLAIN ANALYZE on Query 4.

B. (6 points) Draw the query plan returned using tree and relational algebra notations as given in lecture slides.

C. (2 points) Report the actual runtime of Query 4 now, and compare it with your previous answer in Q4.2(C). Was it worth constructing the indexes?

Hints:

1. Check the statistics collected by PostgreSQL: http://www.postgresql.org/docs/8.4/static/planner-stats.html

2. How to use EXPLAIN command and understand its output: http://www.postgresql.org/docs/8.4/static/sql-explain.html http://www.postgresql.org/docs/8.4/static/performance-tips.html