Homework 4: Query Processing, Query Optimization
(due March 25th, 2014, 3:30pm, in class—hard-copy please)

Reminders:

a. Out of 100 points. Contains 6 pages.
b. Rough time-estimates: 2~4 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: Pranav Nakate.

Q1. Relational Operators [25 points]

Consider the join $R \bowtie_{R.a=S.b} S$ of relation R(a, c) and relation S(b, d) given the following:

- Relation R contains NR = 10,000 tuples with 5 tuples per page.
- Relation S has NS = 200,000 tuples with 20 tuples per page.
- B = 502 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 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 (Pen-n-Paper) [25 points]

In this question, we want to construct the canonical parse trees and analytically compute the selectivities of some queries. Consider the following relational design of a Car Rental Service (the primary key of the relations have been underlined):

Car (cid, brand, crate, description)
Driver (did, dname, address, city, phone)
Rental (cid, did, rentaldate, rentedhours)

The relation Car stores the id of the car, brand of the car, per hour rate of the car and car description such as 'Sports car'. The relation Driver stores the id, name and address information of each driver registered with the Car rental service. The relation Rental stores the information when a driver rents a car. It stores the id of the car, id of the driver, the date when car is rented and the number of hours the car is rented.

Assume that there are 10,000 different cars, 50 different brands, 1 million drivers, and 1 million rental records. Furthermore, we assume there are 1000 cities, the maximum rent is 1000 dollars and the minimum rent is 10 dollars.

Consider the following two queries on the given relations:

Query 1:
```
SELECT did, address
FROM Car as C, Rental as R, Driver as D
WHERE C.cid = R.cid
  AND D.did  = R.did
  AND city = "Blacksburg" AND brand = "Toyota";
```

Query 2:
```
SELECT dname, rentedhours
FROM Car as C, Rental as R, Driver as D
WHERE D.did = R.did
  AND C.cid = R.cid
  AND ((brand = "Honda" AND city = 'Roanoke')
        OR (brand = "Lexus" AND city = 'Roanoke'));
```

Note that the first two equality conditions in the WHERE clause of each query are just the natural join predicates.

Q2.1. (5 points) Describe in plain English what each query does.

Q2.2. (7 points) Query 1 is not hard to parse into a tree.

A. (4 points) Draw the canonical parse tree for Query 1.
Note: Recall that a canonical parse tree has the following properties: (1) it is a left-deep join tree; (2) selections are pushed down as much as possible; and (3) projections are also pushed down as much as possible.

B. (3 points) List the predicates in the tree you got in part A above and compute their selectivities. Show your steps.
Notes: Consider only the non-natural-join predicates. You can refer to formulae on slides for computing selectivities.

Q2.3. (13 points) Query 2 is not easy to parse into a tree. Here is how to do this: First we need to first convert the non-join predicates into so-called Conjunctive Normal Form (CNF). Example:

$(a \text{ AND } b) \text{ OR } c$ in CNF is $(a \text{ OR } c) \text{ AND } (b \text{ OR } c)$

where a, b and c are any Boolean conditions. Hence CNF is essentially a 'AND of ORs'. Note that in contrast the non-join predicates in Query 2 are in 'OR of ANDs' form. Refer to section 14.2.1 on page 445 of the textbook, in case you want to know more.

A. (4 points) Write the equivalent CNF for the non-join predicates of Query 2.

B. (4 points) Now using the CNF form of the non-join predicates, draw the canonical parse tree for Query 2.

C. (5 points) List the predicates in the tree you got in part B above and compute their selectivities. Show your steps. Again consider only the non-join predicates.

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

Consider the following two relations, used in Wikipedia:

- page (page_id,page_title,page_counter,page_touched,page_len)
- page_properties (page_id,property_name,property_value)

Each page is a Wikipedia page; say the Wikipedia page for 'Atlas Shrugged' or 'Accessible Computing'. The page table has the id of the page, the title of the page, the number of times the page is visited, the timestamp of the last page visit and page length. Each page has a property like 'page_image', 'display title' . These properties are present in the page_properties table that has the page id and the property name-value pairs. This table can have multiple page property-value pairs (i.e. multiple tuples) for each page id.

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

We have created sample instances of these two tables on the 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/Spring14/project/postgresql.html

Use the following commands to copy the tables to your private database:

- pg_dump -U YOUR-PID -t page cs4604s14 | psql -d YOUR-PID
- pg_dump -U YOUR-PID -t page_properties cs4604s14 | psql -d YOUR- PID

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

- select count(*) from page; // output – count = 1591065
- select count(*) from page_properties; // output – count = 621166

For this question, it may help to familiarize yourself 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, find the number of attributes of and the number of disk pages occupied by each relation (page and page_properties). Write down the SQL query you used and also paste the output.

Q3.2. (8 points) We want to find the number of distinct values of the 'property_name' attribute of the 'page_properties' table. We will do this via two different ways.

A. (3 points) Write a SQL query that uses the page_properties table to find the number of distinct values of 'property_name'. Paste the output too.

B. (3 points) Now write a SQL query that uses only the 'pg_stats' table instead to find the number of distinct values of 'property_name'. Again, paste the output.

C. (2 points) Compare the outputs from parts A and B above. What do you observe?

Q3.3. (6 points) Consider the following query that retrieves every page that was last visited anytime after Jan 3, 2008. *Note:* You do not need to run anything on the server for this part.

Query 3:
```
SELECT *
FROM page
WHERE page_touched > 20080103083329;
```

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

B. (3 points) Now imagine someone creates a non-clustered B+-Tree index on 'page_touched' attribute. Will it help to speed up the retrieval? Again explain in only 1 line.

Q4. Query Optimization (Joins) [30 points]

Again consider the same two 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 that lists the page title and the page counter for all the pages having certain properties (as given in the WHERE clause).

Query 4:

```
SELECT page_title,page_counter
FROM page a, page_properties b, page_properties c,
        page_properties d
WHERE a.page_id = b.page_id
AND a.page_id =  c.page_id
AND a.page_id = d.page_id
AND b.property_name='wikibase_item'
AND c.property_name='page_image'
AND d.property_name='defaultsort';
```

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) 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 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) Now create an index on 'page_id' attribute of 'page' relation and on the 'page_id' attribute of 'page_properties' relation. Write the two 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