Homework 4: Query Processing, Query Optimization [Solution]  
(due October 16, 2018, 9:30am, in class—hard-copy please)

Reminders:
- Out of 100 points. Contains 6 pages.
- Rough time-estimates: 6~8 hours.
- 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.
- There could be more than one correct answer. We shall accept them all.
- Whenever you are making an assumption, please state it clearly.

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 $NR = 20,000$ tuples with 25 tuples per page.
- Relation $S$ has $NS = 45,000$ tuples with 30 tuples per page.
- $B = 601$ 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 $N$).
Ans:
Smaller Relation – $R$ - 800 pages
Larger Relation – $S$ -1500 pages

Q1.2. (5x4=20 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)
   Ans: I/O (Nested Loops) = $M + pR MN$
   $= 1500 + (30 * 1500 * 800) = 36001500$
where \( M \rightarrow \text{Number of pages in } R; N \rightarrow \text{Number of pages in } S; \)
\( p_R \rightarrow \text{number of tuples per page in } R \)

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

Ans: 
\[
\text{I/O (Block nested Loops)} = M + N\left[\frac{M}{B-2}\right]
\]
\[
= 800 + 1500\left(\frac{800}{599}\right) = 2804
\]

where \( M \rightarrow \text{Number of pages in } R; N \rightarrow \text{Number of pages in } S; \)

\( B \rightarrow \text{No. of buffer pages} \)

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

Ans: 
\[
\text{I/O} = 2M(\log M/\log B) + 2N(\log N/\log B) + M + N = 3429 + 1672 = 5101
\]

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

Ans: In partitioning phase, 
\[
\text{I/O} = 2(M+N) = 2(800+1500) = 4600
\]
In matching phase, 
\[
\text{I/O} = M+N = 800+1500 = 2300
\]
Total I/O = 6900

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).
ANS: 10 values (500 stores, 50 store locations, uniform distribution of values)

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)
ANS: selectivity (T.size>7 AND I.quantity = 20) = 0.7 * (20/1000) * 50000
= 700

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).

ANS:
For tree(ii):

I/O (Nested Loops) = M + pMN = 25 + 25*40*50 = 50025

I/O (Sort-Merge) = 2M(logM/logB) + 2N(logN/logB) + M + N = 3429+1672 = 204.96

I/O (Block nested Loops) = M + N[M/(B-2)] = 25 + 50 * (25/68) = 43.4
I/O(Hash-Join) = 3(M+N) = 3(25+50) = 225

Q3. Query Optimization (Stats and Range Queries) [20 points]
Consider the following two relations, used for book rating:
- book (ISBN, title, author, year, publisher, image_s, image_m, image_l)
- users (id, location, age)

The above information was gathered in a 4-week crawl from the book-crossing.com community website where users rate books. The book table has ‘ISBN’, ‘book title’, ‘book author’, ‘Year of publication’, ‘Publisher’, ‘Image-URL-S’ which is the URL for small size image of the book (like a thumbnail), ‘Image-URL-M’ which indicates the URL for medium size image, and ‘Image-URL-L’ which is the URL for the large size image; The content of ‘book’ table was obtained from Amazon Web Service. In the ‘user_book_rating’ table we have the ‘user_id’, which is the id of the user who rates a book, ‘ISBN’ of the book and the ‘book rating’ of the user for the book. Each user can rate multiple books. The ‘users’ table has the ‘id’, ‘location’ and ‘age’ of the users who rate the books.

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/Fall18/project/postgresql.html

Use the following commands to copy the tables to your private database:
- `pg_dump -U YOUR-PID -t book cs4604f18 | psql -d YOUR-PID`
- `pg_dump -U YOUR-PID -t user_book_rating cs4604f18 | psql -d YOUR-PID`
- `pg_dump -U YOUR-PID -t users cs4604f18 | psql -d YOUR-PID`

**Sanity Check:** run the following two statements and verify the output.
- `select count(*) from book;        // output – count = 250012`
- `select count(*) from user_book_rating;       // output – count = 1149769`
• select count(*) from users;  // output – count = 278683

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 ‘book’, ‘user_book_rating’ and ‘users’ 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.

Ans: select relname, reltuples, relhaspkey from pg_class where relname in ('book', 'users', 'user_book_rating');

<table>
<thead>
<tr>
<th>relname</th>
<th>reltuples</th>
<th>relhaspkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>book</td>
<td>250012</td>
<td>f</td>
</tr>
<tr>
<td>users</td>
<td>278682</td>
<td>f</td>
</tr>
<tr>
<td>user_book_rating</td>
<td>1.14977e+06</td>
<td>f</td>
</tr>
</tbody>
</table>

Q3.2. (8 points) We want to find the top three years in which most publications happen. There are two ways to do that:

A. (4 points) Write a SQL query that uses the book table find the top three years in which most publications happen. Paste the output too when you run the query.

ANS: select year, count(*) as countbooks from book group by year order by countbooks DESC LIMIT 3;

<table>
<thead>
<tr>
<th>year</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2002</td>
<td></td>
</tr>
<tr>
<td>2001</td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td></td>
</tr>
</tbody>
</table>

3 rows
B. (4 points) Now write a SQL query that uses only the ‘pg_stats’ table instead to find the top three years in which most publications happen. Again, paste the output.

ANS: 
```
SELECT (most_common_vals::varchar::varchar)[1:3] FROM pg_stats
WHERE tablename='book' and attname='year' LIMIT 10;
```

Note: to get the first three elements of a ‘list’ type (e.g. ‘mylist’), you can use ‘mylist[1:3]’, and to get the first three elements of ‘anyarray’ type (e.g. ‘myarray’), you can use (myarray::varchar::varchar)[1:3] 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 books published after 1880. Note: You do not need to run anything on the server for this part.

Query 3: 
```
SELECT *
FROM book
WHERE year > 1880;
```

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.
   Ans: The optimizer will use a simple file sequential scan since there is no index.

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.
   Ans: Not necessarily help speed up the retrieval since it will depend on the way the year is distributed across the table.

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?

245247

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

ANS:

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

AND users.age = 20
AND user_book_rating.book_rating = 7;

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?
   Ans: Gives us information about books, and their ratings for all books published after year 2000, reviewed by students who are 20 years old and having a book rating of 7.

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

-----------------------------------------------
--------------------------------------------------------------------------------
Hash Join (cost=38247.42..106658.90 rows=309 width=62) (actual time=848.520..1357.706 rows=172 loops=1)
  Hash Cond: (book.isbn = user_book_rating.isbn)
  -> Seq Scan on book (cost=0.00..65628.15 rows=67400 width=79) (actual time=0.024..428.008 rows=67447 loops=1)
    Filter: (year >= 2000)
  -> Hash (cost=38233.07..38233.07 rows=1148 width=25) (actual time=848.322..848.322 rows=612 loops=1)
    -> Seq Scan on user_book_rating (cost=0.00..25118.11 rows=77073 width=46) (actual time=65.541..652.272 rows=76456 loops=1)
    Filter: (book_rating = 7)
  -> Hash Join (cost=12525.42..38233.07 rows=1148 width=25) (actual time=156.182..846.663 rows=612 loops=1)
    Hash Cond: (user_book_rating.user_id = users.id)
    -> Seq Scan on user_book_rating (cost=0.00..25118.11 rows=77073 width=46) (actual time=65.541..652.272 rows=76456 loops=1)
      Filter: (book_rating = 7)
    -> Hash (cost=12473.52..12473.52 rows=4152 width=21) (actual time=90.449..90.449 rows=4046 loops=1)
      -> Seq Scan on users (cost=0.00..12473.52 rows=4152 width=21) (actual time=0.043..84.063 rows=4046 loops=1)
        Filter: (age = 20)
Total runtime: 1358.111 ms
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).

Ans:
- book → ISBN,, year,
• user_book_rating → user_id, book_rating
• users → id, age

create index yearindex on book (year);
create index useridindex on user_book_rating (user_id);
create index bookratingindex on user_bookrating (book_rating);
create index ageindex on users (age);

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.

QUERY PLAN

---------------------------------------------
-----------------------------------------------
-----------------------------------------------------------------
Nested Loop (cost=8893.03..28811.91 rows=302 width=63) (actual
time=52.609..317.498 rows=172 loops=1)
  > Hash Join (cost=8893.03..21151.08 rows=1121 width=25) (actual
time=50.876..300.473 rows=612 loops=1)
    Hash Cond: (user_book_rating.user_id = users.id)
  > Bitmap Heap Scan on user_book_rating (cost=1405.95..13089.98 rows=
75042 width=46) (actual time=13.353..165.598 rows=76456 loops=1)
    Recheck Cond: (book_rating = 7)
  > Bitmap Index Scan on bookratingindex (cost=0.00..1387.19 rows=
75042 width=0) (actual time=11.479..11.479 rows=76456 loops=1)
    Index Cond: (book_rating = 7)
  > Hash (cost=7435.05..7435.05 rows=4162 width=21) (actual time=37.41
4..37.414 rows=4046 loops=1)
  > Bitmap Heap Scan on users (cost=80.54..7435.05 rows=4162 wid
th=21) (actual time=0.989..31.456 rows=4046 loops=1)
Recheck Cond: (age = 20)
-> Bitmap Index Scan on ageindex  (cost=0.00..79.50 rows=4
162 width=0) (actual time=0.596..0.596 rows=4046 loops=1)
Index Cond: (age = 20)
-> Index Scan using isbnindex on book  (cost=0.00..6.82 rows=1 width=80)
(actual time=0.024..0.024 rows=0 loops=612)
Index Cond: (book.isbn = user_book_rating.isbn)
Filter: (book.year >= 2000)
Total runtime: 317.781 ms
(16 rows)

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?
Ans: Actual runtime of query 4 – 318 ms. It was worth constructing indexes, the time has fallen to \( \frac{1}{4} \) almost.

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