Assignment 2 - Sample Solution

1. [13.1] Consider the following SQL query for our bank database:

   ```
   select T.branch-name
   from branch T, branch S
   where T.assets > S.assets and S.branch-city = Brooklyn
   ```

Write an efficient relational-algebra expression that is equivalent to this query. Justify your choice.

**Answer:**

\[
\Pi_{T.branch-name}((\Pi_{branch-name,assets}(\rho_T(branch))) \bowtie T.assets > S.assets
\]

\[
(\Pi_{assets}(\sigma_{branch-city='Brooklyn'})(\rho_S(branch)))
\]

This expression performs the theta join on the smallest amount of data possible. It does this by restricting the right hand side operand of the join to only those branches in Brooklyn, and also eliminating the unneeded attributes from both the operands.

2. [13.3] Let relations \( r_1(A, B, C) \) and \( r_2(C, D, E) \) have the following properties: \( r_1 \) has 20,000 tuples, \( r_2 \) has 45,000 tuples, 25 tuples of \( r_1 \) fit on one block, and 30 tuples of \( r_2 \) fit on one block. Estimate the number of block accesses required, using each of the following join strategies for \( r_1 \bowtie r_2 \):

   (a) Nested-loop join
   (b) Block nested-loop join
   (c) Merge join
   (d) Hash join

**Answer:**

\( r_1 \) needs 800 blocks, and \( r_2 \) needs 1500 blocks. Let us assume \( M \) pages of memory. If \( M > 800 \), the join can easily be done in 1500 + 800 disk accesses, using even plain nested-loop join. So we consider only the case where \( M \leq 800 \) pages.

   (a) Nested-loop join:
   Using \( r_1 \) as the outer relation we need \( 20000 \cdot 1500 + 800 = 30,000,800 \) disk accesses, if \( r_2 \) is the outer relation we need \( 45000 \cdot 800 + 1500 = 36,001,500 \) disk accesses.

   (b) Block nested-loop join:
   If \( r_1 \) is the outer relation, we need \( \lceil \frac{800}{M-1} \rceil \cdot 1500 + 800 \) disk accesses, if \( r_2 \) is the outer relation we need \( \lceil \frac{1500}{M-1} \rceil \cdot 800 + 1500 \) disk accesses.
(c) Merge-join:
Assuming that \( r_1 \) and \( r_2 \) are not initially sorted on the join key, the total sorting cost inclusive of the output is
\[
B_s = 1500 \cdot (2\lceil \log_M (1500/M) \rceil + 2) + 800 \cdot (2\lceil \log_M (800/M) \rceil + 2)
\]
disk accesses. Assuming all tuples with the same value for the join attributes fit in memory, the total cost is \( B_s + 1500 + 800 \) disk accesses.

(d) Hash-join:
We assume no overflow occurs. Since \( r_1 \) is smaller, we use it as the build relation and \( r_2 \) as the probe relation. If \( M > 800/M \), i.e. no need for recursive partitioning, then the cost is \( 3(1500 + 800) = 6900 \) disk accesses, else the cost is \( 2(1500 + 800)\lceil \log_M (800) - 1 \rceil + 1500 + 800 \) disk accesses.

3. [13.6] Suppose that a \( B^+ \)-tree index on branch-city is available on relation branch, and that no other index is available. List different ways to handle the following selections that involve negation:

- (a) \( \sigma_{\neg (\text{branch-city} < 'Brooklyn')} \) (branch)
- (b) \( \sigma_{\neg (\text{branch-city} = 'Brooklyn')} \) (branch)
- (c) \( \sigma_{\neg (\text{branch-city} < 'Brooklyn' \lor \text{assets} < 5000)} \) (branch)

**Answer:**

(a) Use the index to locate the first tuple whose branch-city field has value “Brooklyn”. From this tuple, follow the pointer chains till the end, retrieving all the tuples.

(b) For this query, the index serves no purpose. We can scan the file sequentially and select all tuples whose branch-city field is anything other than “Brooklyn”.

(c) This query is equivalent to the query
\[
\sigma_{(\text{branch-city} \geq 'Brooklyn' \land \text{assets} \geq 5000)} \) (branch)
\]
Using the branch-city index, we can retrieve all tuples with branch-city value greater than or equal to “Brooklyn” by following the pointer chains from the first “Brooklyn” tuple. We also apply the additional criteria of assets \( \geq 5000 \) on every tuple.

4. [13.13] The hash join algorithm as described in Section 13.5.5 computes the natural join of two relations. Describe how to extend the hash join algorithm to compute the natural left outer join, the natural right outer join and the natural full outer join. (Hint: Keep extra information with each tuple in the hash index, to detect whether any tuple in the probe relation matches the tuple in the hash index.) Try out your algorithm on the customer and depositor relations.

**Answer:** This will be provided separately (hopefully!).

5. Consider a disk with average seek time of \( 12 \mu \)s, average latency of \( 6 \mu \)s, and a page transfer time of \( 2 \mu \). You are given 320 buffer pages and asked to sort a file of 10,000,000 pages. The initial pass will create runs of 320 pages each. Evaluate the cost of the following approaches for the subsequent merging passes:

(a) Do \( 319 \) way merges.
(b) Create 256 input buffers of 1 page each and one output buffer of 64 pages, and do 256-way merges.
(c) Create 16 input buffers of 16 pages each and one output buffer of 64 pages, and do 16-way merges.
(d) Create 8 input buffers of 32 pages each and one output buffer of 64 pages, and do 8-way merges.
(e) Create 4 input buffers of 64 pages each and one output buffer of 64 pages, and do 4-way merges.

**Answer:**

After the initial pass, there will be \( \lfloor \frac{10,000,000}{320} \rfloor = 31250 \) runs.

(a) We need \( \lceil \log_{319}(31250) \rceil = 2 \) passes. The time per pass is

\[
10,000,000 \cdot 2 \cdot (12 + 6 + 2)\mu = 400,000s.
\]

Total time is 2 * 400,000 = 800,000 seconds.

(b) Number of passes: \( \lceil \log_{256}(31250) \rceil = 2 \). Read time per pass will be

\[
10,000,000 \cdot (12 + 6 + 2)\mu = 200,000s
\]

and the write time will be

\[
\lfloor \frac{10,000,000}{64} \rfloor \cdot (12 + 6 + 2 \cdot 64)\mu = 156250 \cdot 146 = 22,812,500\mu = 22,812.5s
\]

Total time is 2 * (200,000 + 22,812.5) = 445,625 seconds.

(c) Number of passes: \( \lceil \log_{16}(31250) \rceil = 4 \).
Read time: \( \lfloor \frac{10,000,000}{16} \rfloor \cdot (12 + 6 + 2 \cdot 16)\mu = 625,000 \cdot 50\mu = 31,250 \) seconds.
Write time: as above, 22,812.5 seconds.
Total time: 4 * (31,250 + 22,812.5) = 216,250 seconds.

(d) Number of passes: \( \lceil \log_{8}(31250) \rceil = 5 \).
Read time: \( \lfloor \frac{10,000,000}{32} \rfloor \cdot (12 + 6 + 2 \cdot 32)\mu = 312,500 \cdot 82\mu = 25,625 \) seconds
Write time: 22,812.5 seconds.
Total time: 5 * (25,625 + 22,812.5) = 242,187.5 seconds.

(e) Number of passes: \( \lceil \log_{4}(31250) \rceil = 8 \).
Read time: same as write time, 22,812.5 seconds.
Write time: 22,812.5 seconds.
Total time: 8 * 2 * 22,812.5 = 365,000 seconds.