CS386D Problem Set #1

[1] Consider the following query:

```sql
select * from A a, B b, C c, D d
```

(a) List all of the logical access plans are examined by the System R optimizer. Hint: do not show the stream ordering and join predicate parameters in your expressions. Follow the analysis in the class notes (choose a sink and find all 1-relation queries, then prune, 2-relation queries, then prune, etc.)

(b) What logical access plans are not examined by the System R optimizer? Why are they not considered?

[2] Consider a linear query graph. What is the size of the search space that System R examines? (or how many plans does System R generate)? Pick one question — they have different answers.

[3] Consider the following attributes, their cardinalities, and index storage structures:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Cardinality</th>
<th>Storage Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20</td>
<td>B+ trees</td>
</tr>
<tr>
<td>B</td>
<td>2000</td>
<td>B+ trees</td>
</tr>
<tr>
<td>C</td>
<td>2000</td>
<td>hash</td>
</tr>
<tr>
<td>D</td>
<td>20</td>
<td>Not Indexed</td>
</tr>
</tbody>
</table>

Now consider the following local predicates. For each predicate, what index would you use (if any) to most efficiently retrieve the tuples that satisfy this predicate:

(a) B=3 or B=4
(b) B=66 and C=12
(c) B>3 and C>77
(d) B=22 and A = 15
(e) D=44 and B>34

[4] Suppose join predicates are of the form “A or B or C or ...” where A, B, C, ... are typical conjunctive join predicates. How would you generalize the System R algorithm to process such queries?
solution

[1a]

1 relation queries are a, b, c, d
2 relation queries are a-b, a-d, b-a, b-c, c-b, c-d, d-a, d-c
pruning: \( ab = \min(a-b, b-a), bc = \min(b-c, c-b), cd = \min(c-d, d-c) \), \( ad = \min(a-d, d-a) \)

3 relation queries are: ab-c, ab-d, bc-a, bc-d, cd-a, cd-b, ad-b, ad-c
pruning: \( abc = \min(ab-c, bc-a), abd = \min(ab -d, ad-b), bcd = \min( bc-d, cd-b), acd = \min( cd-a, ad-c ) \)

4 relation queries are: abc-d, adb-c, bcd-a, acd-b
pruning abcd = \( \min( abc-d, adb-c, bcd-a, acd-b ) \)

[1b] system r produces left-deep operator trees (meaning that the right operation is a retrieval, never a join). So a plan never considered is \(((a,b),(c,d))\) i.e., \( \text{join(join(a,b), join(c,d)))} \)

[2] A linear query of n relations is a query graph that is a line:

```
1 2 3    n-1 n
```

How many distinct logical access plans (or equivalently, join orderings) could be produced by the System R algorithm for a linear query of n relations? You are to ignore stream orderings and simply consider the number of distinct orders in which relations can be joined. Define a (closed-form) formula for \( S(n) \). You may find the following identity helpful:

\[
2^k = \sum_{i=0}^{k} \binom{k}{i}
\]

Let node \( i \) be the sink node. There are \( i-1 \) nodes to the “left” of \( i \) and \( n-i \) nodes to the right. There are \( \binom{n-1}{i-1} \) ways of forming a logical access plan, given node \( i \) as a sink. Reason: fixing \( i \), nodes can be dragged down the sink in any order of listing from right to left. Summing over all positions for \( i \), we have the total number of logical plans that can be created:

\[
S(n) = \sum_{i=1}^{n} \binom{n-1}{i-1}
\]

It follows that \( S(n) = 2^{n-1} \).

Another way to interpret this question is how many plans are actually generated (i.e., taking into account pruning). The number of plans for 1 relation is \( n \). The number of plans for 2 relations (before pruning) is approx \( n-1+n-1= 2^{*(n-1)} \). Note: for a line of \( n \) nodes, only \( n-1 \) nodes can be joined with a
node to the right, and only n-1 nodes can be joined to the left. The number of plans for 3 relations is \(n-2+n-2=2(n-2)\). For i relations, there are \(2(n-i+1)\) plans. Summing, the complexity is \(O(n^2)\).

[3] Consider the following attributes, their cardinalities, and index storage structures:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Cardinality</th>
<th>Storage Structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>20</td>
<td>B+ trees</td>
</tr>
<tr>
<td>B</td>
<td>2000</td>
<td>B+ trees</td>
</tr>
<tr>
<td>C</td>
<td>2000</td>
<td>hash</td>
</tr>
<tr>
<td>D</td>
<td>20</td>
<td>Not Indexed</td>
</tr>
</tbody>
</table>

Now consider the following local predicates. For each predicate, what index would you use (if any) to most efficiently retrieve the tuples that satisfy this predicate:

(a) B=3 or B=4 -- either scan or use B index twice
(b) B= 66 and C=12 -- C would be fastest (if you use 1 index). You could use multiple indices and take the intersection of their pointers.
(c) B>3 and C>77 -- scan
(d) B=22 and A = 15 -- use B index (could intersect lists, but this is not clear that even creating an index for A is that useful).
(e) D=44 and B>34 -- scan

[4] There are a variety of answers that you could postulate. The “framework” of System-R is extraordinarily robust. Just as a join predicate (A.x = B.y and A.z=B.w) could be supplied as an argument to a join operation (e.g., JOIN(A,B, A.x=B.y and A.z=B.w)), there’s no reason why (A.x=B.y or A.z=B.w) could be provided as an argument to a join operation (e.g., JOIN(A,B, A.x=B.y or A.z=B.w)). The trick here is what algorithm could you use to process this join predicate. Nested loops would work just fine. As a possible future problem, is there a reasonable generalization of merge-join and/or hash-join to deal with such join predicates?

You could have other, more drastic solutions: you could allow cross-product edges with join predicate labels. They would be considered first, before pure or unrestricted cross products.

There is even a rather simple generalization of System-R algorithm to allow an additional operation that takes a stream S and predicate P (join predicate, relation predicate, mix of the two) and produces a stream where only records of S that satisfy P are output.

There is no end to the creativity of how this could be accomplished. If you don’t see such possibilities, as I list above, please (by all means) ask in class. If you do understand my points, you have a very good understanding of this material.