B561 Assignment 6:
Transactions, Query Processing, and Recovery Solutions
C.M. Wyss
December 10, 2001

1. Exercise 12.4 in your text (page 356).

Solution:

1. \(|S| + |S| \cdot |R| = 200 + 200 \cdot 1000 = 200,200\) page accesses. We need at least 3 buffer pages to achieve this (one for \(R\), one for \(S\) and one for the output).

2. \(|S| + |R| \cdot \left\lceil \frac{|S|}{B-2} \right\rceil = 200 + 1000 \cdot \left\lceil \frac{200}{49} \right\rceil = 4,200\) page accesses. When \(B < 52\) then \(\frac{200}{49} \cdot B > 4\), so we need at least 52 pages to achieve the 4,200 figure.

3. We have that the number of buffers \(52 > \sqrt{|R|} = \sqrt{1,000} \approx 32\). Thus, the total cost of sort-merge join is \(3 \cdot (|S| + |R|) = 3 \cdot (1,000 + 200) = 3,600\). We would need at least 32 buffer pages to achieve this.

4. The cost is \(3 \cdot (|S| + |R|) = 3 \cdot (1,000 + 200) = 3,600\) I/Os. We need approximately \(B > \sqrt{f \cdot |S|} \approx \sqrt{1.5 \cdot |S|} = 21\) buffer pages to achieve this (assuming \(f\) is close to 1.5, i.e. the data is pretty uniformly distributed).

5. Let’s say there are \(K\) distinct \(a\) values in the \(R\) relation where \(K\) is small enough that we can hold an in-memory hash table for \(R\) on attribute \(a\) (\(K = 500\) would work if the data is well-behaved). Then we can build this hash table in one scan of \(R\), and read \(S\) only once to probe the hash table. This gives an optimal I/O cost of \(|R| + |S| = 1,000 + 200 = 1,200\) page I/Os. We can’t get any better than this, since we’d better at least look at both relations.

6. The join produces at most \(|R| \cdot |S| = 1,000 \cdot 200 = 200,000\) tuples. We’d need to buffer one page at a time to write to disk, so we only need 1 more buffer page for this.

7. Sort-merge join (part 3) will probably be faster, as will hash join (part 4). However, our “optimal” strategy (part 5) won’t work, since \(a\) will have 1,000 distinct values. So we’d need at least 100 pages to hold the hash table for \(R\).
2. Exercise 14.8 in your text (page 409).

Solution:

1. At a minimum, we need to know the sizes of the relations Suppliers, Supply and Parts, and whether we have indices on the pid, sid, price and city attributes of the appropriate relations. Also, if there is an index, we need to know the type. Additionally, histogram information on the distribution of the data in these columns would help us to estimate the sizes of intermediate results.

2. A System R-type optimizer will consider all left-deep join plans as on pages 393-395. There are six such trees (depicted in figure 1).

![Image of join orderings]

Figure 1: Left-deep join orderings considered.

3. A B+-tree index on attribute price of relation Parts would be very helpful, since B+-trees are best at range queries. A hash index would be good on attribute cite of relation Suppliers, so we can perform the selection \( \sigma_{\text{city} = \text{"Madison"}}(\text{Suppliers}) \) optimally. In addition, indices (of either type) on the sid attributes of relations Suppliers and Supply and on the pid attributes of relations Supply and Parts would noticeably expedite the joins.

3. Consider the following SQL query

```sql
SELECT * FROM Suppliers
WHERE city = 'Madison'
```
SELECT ROADID
FROM ROADS R, ZONES Z1, ZONES Z2
WHERE R.SRCZONE = Z1.ZONEID AND R.ENDZONE = Z2.ZONEID AND
    Z1.TYPE = 'R' AND Z2.TYPE = 'C' AND R.DIST < 10

(a) Translate this query into an RA expression, using the naive translation algorithm given in class.

Solution:

$$\Pi_{\text{ROADID}}(\sigma_{\text{SRCZONE} = \text{ZONEID}_1 \wedge \text{ENDZONE} = \text{ZONEID}_2 \wedge 
    \text{TYP}_1 = 'R' \wedge 
    \text{TYP}_2 = 'C' \wedge 
    \text{DIST} < 10} (\text{ROADS} \times \rho_{X \rightarrow X_1} (\text{ZONES}) \times \rho_{X \rightarrow X_2} (\text{ZONES})))$$

(b) Use the rule-based method developed in class to transform the RA expression you obtained in (a) into an optimized RA expression. If you make any assumptions (e.g. about the size of the relations involved), state these clearly.

Solution: We assume the ZONES relation is significantly smaller than the ROADS relation. See figure 2 for the optimization.

Figure 2: Rule-Based Optimization of Query.
4. Exercize 18.4 in your text (page 539).

Solution:

1. The serializability graph for this schedule is shown in figure 3.

```
Figure 3: Serializability graph.
```

2.(a) T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y), T3:commit, T2:commit

2.(b) T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y), T2:commit, T3:abort

2.(c) Since there is a cycle in the serializability graph, there is no way to make this schedule conflict serializable without deleting actions (or inserting commit/abort in the middle of a transaction, which has the same effect).

5. Discuss how the recovery algorithm given in class changes if there is no available cache. For example, is a log still required in this situation?

Solution: A log is still required, as we may crash at any time and the changes of uncommitted transactions will need to be undone. The net effect of no cache is that every write is a write-through, meaning all changes hit the disk immediately and cannot grow stale in the cache. Thus, we do not need to redo any actions in our recovery management (i.e. recovery effectively becomes a NO-REDO/UNDO situation, see the solutions to 1999 assignment 6, question 6). Also, we cannot assume the cache in our RM pseudo-code (this is only a slight modification).

6. Rewrite the UNDO/REDO recovery algorithm given in class assuming no transactions can abort. Assume no deadlock occurs (so you only need consider system failures).

Solution: Basically, just ignore (delete) the abort list. This question probably should have been written differently. Oh well.