Query Processing – Practice Questions Solution

1. Consider these relations with the following properties:

r(A, B, C)	s(C, D, E)
30,000 tuples	60,000 tuples
25 tuples fit on 1 block	30 tuples fit on 1 block

a) Estimate the number of disk block accesses required for a natural join of r and s using a nested-loop join if r is used as the outer relation.

r requires (30000/25) 1200 blocks of storage and s requires (60000/30) 2000 blocks of storage. Hence, using the formula given on page 399 of the text (n_r x b_s + b_r), 30000*2000+1200 = 60,001,200 disk accesses are required for a nested loop join.

b) Estimate the number of disk block accesses required for a natural join of r and s using a block nested-loop join if s is used as the outer relation. Assume that there are more than 2000 memory buffers available to facilitate this operation, where each memory buffer can buffer one disk block.

r requires b_r = (30000/25) 1200 blocks of storage and s requires b_s = (60000/30) 2000 blocks of storage. Hence, using the formula given on page 400 of the text $\lceil b_s/M-1 \rceil * b_r + b_s$, $\lceil 2000/M-1 \rceil$ * 1200 + 2000 = 3200 disk block accesses are required for a block nested-loop join.

Note that the question is set up in such a way to enable the best-case scenario (*i.e.* enough buffers to cache the s relation) for a block-nested loop join. If instead the worst case is identified, the part marks would be given for the worst-case formula (b_r x b_s + b_r) and for identification of b_r = 1200 and b_s = 2000.

2. Consider the following SQL query on the schema *branch(branch_name, branch_city, assets)*:

select t.branch_name
from branch t, branch s
where t.assets > s.assets and s.branch_city = 'Burnaby';

Write an efficient relational algebra expression that is equivalent to this query and JUSTIFY your choice with an explanation.

$$\Pi_{T.branch\_name}((\Pi_{branch\_name,\ assets}\ (\rho_T\ (branch)))\ \bowtie_{T.assets\ >\ S.assets}$$
$$(\Pi_{assets}\ (\sigma_{(branch\_city\ =\ 'Burnaby')}\ (\rho_S\ (branch)))))$$

This expression is efficient because:

- performs the theta join on the smallest amount of data possible. It restricts the right-hand side of the join to only those branches in 'Burnaby'
- eliminates the unneeded attributes from both the operands

3. Suppose we have the following relations:

   employee(emp_id, salary, age, dept_id)
   department(dept_id, budget, status)

   Each *employee* record is 20 bytes long and each *department* record is 40 bytes long. There are 20,000 tuples in the *employee* table and 5000 tuples in the *department* table. The *dept_id* attribute in *employee* is a foreign key of the *department* relation. The file system supports a page size of 4000 bytes and there are 12 buffer pages available to the database. Assume we are using the number of page I/O's as the measure of a query's cost. The following indices exist:

   - a clustering (*i.e.* primary) index on the *dept_id* attribute in *employee,*
   - a non-clustering (*i.e.* secondary) index on the *age* attribute in *employee,*
   - a clustering index on the *dept_id* attribute in *department*

a) Consider the SQL query

```
select * from employee where age > 30
```

Let N = the number of tuples retrieved with this query. For what values of N would a sequential table scan of the *employee* relation be cheaper than processing the query using the index? **Explain** your answer.

Given that there are 20000 tuples of 20 bytes, the *employee* relation occupies 100 pages of 4000 bytes apiece. To retrieve N tuples using the secondary index on *age*, N page I/O's are required. To perform a full table scan of the *employee* relation would require 100 page I/O's, so if we expect more than 100 tuples to be returned by the query, it would be cheaper to do a full table scan than to use the secondary index.

b) Consider the SQL query

```
select *
from employee, department
where employee.dept_id = department.dept_id
```

What evaluation plan would a query optimizer likely choose to get the least estimated cost?

The relational algebra expression for this query is:

$$\text{employee} \bowtie \text{department}$$

Chapter 12 of the text covers a number of different algorithms for performing joins and cites cases in which each algorithm should be used. The clustering indices on dept_id in both employee and department indicate that both relations are sorted on the join attribute. Thus, it would be cheapest to use the merge-join algorithm, which gives a total cost of $b_{employee} + b_{department}$ page I/O's (100 + 50).