Week 9 Exercises: Query Processing with Relational Operations (part 2) Solutions

Answer 14.1 The answer to each question is given below.

1. Hybrid hash join improves performance by comparing the first hash buckets during the partitioning phase rather than saving it for the probing phase. This saves us the cost of writing and reading the first partition to disk.
2. Hash join provides good performance for equality joins, and can be tuned to require very few extra disk accesses beyond a one-time scan (provided enough memory is available). However, hash join is sensitive to data skew and it is not applicable for non-equality joins.
   Sort-merge join is less sensitive to data skew. Sort-merge also leaves the results sorted which is often desired. Sort-merge join has extra costs when you have to use external sorting (there is not enough memory to do the sort in-memory).
   Block nested loops join works for theta joins.
3. If the join condition is not equality, you can use sort-merge join, index nested loops (if you have a range style index such as a B+ tree index or ISAM index), or block nested loops join. Hash joining works best for equality joins and is not suitable otherwise.

Answer 14.4 Let M = 1000 be the number of pages in R, N = 200 be the number of pages in S, and B = 52 be the number of buffer pages available.

1. Basic idea is to read each page of the outer relation, and for each page scan the inner relation for matching tuples. Total cost would be
   $$pagesinouter + (pagesinouter * pagesininner)$$
   which is minimized by having the smaller relation be the outer relation.
   $$TotalCost = N + (N * M) = 200,200$$
   The minimum number of buffer pages for this cost is 3.
2. This time we read the outer relation in *blocks*, and for each block scan the inner relation for matching tuples. So the outer relation is still read once, but the inner relation is scanned only once for each outer block, of which there are $\lceil \frac{pagesinouter}{B-2} \rceil = \lceil \frac{200}{50} \rceil = 4$.
   $$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 4,200$$

   If the number of buffer pages is less than 52, the number of scans of the inner would be more than 4 since$\lceil \frac{200}{49} \rceil = 5$. The minimum number of buffer pages for this cost is therefore 52.
3. The cost of Hash Join is $3 * (M + N)$ if $B > \sqrt{f * N}$ where f is a 'fudge factor' used to capture the small increase in size involved in building a hash table, and N is the number of pages in the smaller relation, S. Since $\sqrt{N} \approx 14$, we can assume

that this condition is met. We will also assume uniform partitioning from our hash function.
$TotalCost = 3 * (M + N) = 3,600$
Without knowing f we can only approximate the minimum number of buffer pages required.

4. The optimal cost would be achieved if each relation was only read once. We could do such a join by storing the entire smaller relation in memory, reading in the larger relation page-by-page, and for each tuple in the larger relation we search the smaller relation (which exists entirely in memory) for matching tuples. The buffer pool would have to hold the entire smaller relation, one page for reading in the larger relation, and one page to serve as an output buffer.
   $TotalCost = M + N = 1,200$
   The minimum number of buffer pages for this cost is N + 1 + 1 = 202.
5. Any tuple in R can match at most one tuple in S because *S.b* is the primary key (which means the S.b field contains no duplicates). So the maximum number of tuples in the result is equal to the number of tuples in R, which is 10,000.
   The size of a tuple in the result could be as large as the size of an R tuple plus the size of an S tuple (minus the size of the shared attribute). This may allow only 5 tuples to be stored on a page. Storing 10,000 tuples at 5 per page would require 2000 pages in the result.

Answer 14.5.

1. Block Nested Loops Join:
   Having R as an outer relation:
   $$TotalCost = R + S * \lceil \frac{R}{B-2} \rceil = 1000 + 100 * \lceil \frac{1000}{12-2} \rceil = 11,000$$
   Having S as an outer relation:
   $$TotalCost = S + R * \lceil \frac{S}{B-2} \rceil = 100 + 1000 * \lceil \frac{100}{12-2} \rceil = 10,100$$
   Therefore, we choose the option with the lower cost (having S as an outer relation).

2. Hash Join:
   Partitioning phase: 2 * (R + S) = 2 * (1000 + 100) =2200
   Probing (matching) phase: (R + S) = 1100
   Total IO cost = 3300 IOs.
3. Sort-Merge Join

   Sort R: # of passes = 1+ $\lceil \log_{B-1} \lceil R/B \rceil \rceil$; Cost = 2 * R * (# of passes);

   Cost = 2 * 1000 * (1+ $\lceil \log_{11} \lceil 1000/12 \rceil \rceil$) = 6000 IOs.

   Sort S: # of passes = 1+ $\lceil \log_{B-1} \lceil S/B \rceil \rceil$; Cost = 2 * S * (# of passes);

   Cost = 2 * 100 * (1+ $\lceil \log_{11} \lceil 100/12 \rceil \rceil$) = 400 IOs.

   Merge: R + S = 1000 + 100 = 1100 IOs.
   Total IO cost 6000 + 400 + 1100 = 7500$ IOs.