Assignment 4: Part I - Sample Solution

1. [13.11 & 13.12] Design a variant of the hybrid merge-join algorithm for the case where both relations are not physically sorted, but both have a sorted secondary index on the join attributes. Estimate the number of block accesses required by your solution to Exercise 13.6 for \( r_1 \Join r_2 \), where \( r_1 \) and \( r_2 \) are as defined in Exercise 13.5.

**Answer:**
We merge the leaf entries of the first sorted secondary index with the leaf entries of the second sorted secondary index. The result file contains pairs of addresses, the first address in each pair pointing to a tuple in the first relation, and the second address pointing to a tuple in the second relation.

This result file is first sorted on the first relations addresses. The relation is then scanned in physical storage order, and addresses in the result file are replaced by the actual tuple values. Then the result file is sorted on the second relations addresses, allowing a scan of the second relation in physical storage order to complete the join.

\( r_1 \) occupies 800 blocks, and \( r_2 \) occupies 1500 blocks. Let there be \( n \) pointers per index leaf block (we assume that both the indices have leaf blocks and pointers of equal sizes). Let us assume \( M \) pages of memory, \( M < 800 \). \( r_1 \)'s index will need \( B_1 = \lceil \frac{20000}{n} \rceil \) leaf blocks, and \( r_2 \)'s index will need \( B_2 = \lceil \frac{45000}{n} \rceil \) leaf blocks. Therefore the merge join will need \( B_3 = B_1 + B_2 \) accesses, without output. The number of output tuples is estimated as \( n_o = \lceil \frac{20000}{\max(V(C,r_1),V(C,r_2))} \rceil \). Each output tuple will need two pointers, so the number of blocks of join output will be \( B_{o1} = \lceil \frac{n_o}{n/2} \rceil \). Hence the join needs \( B_j = B_3 + B_{o1} \) disk block accesses.

Now we have to replace the pointers by actual tuples. For the first sorting,

\[ B_{s1} = B_{o1}(2\lceil \log_{M-1}(B_{o1}/M) \rceil + 2) \]

disk accesses are needed, including the writing of output to disk. The number of blocks of \( r_1 \) which then have to be accessed in order to replace the pointers with tuple values is \( \min(800, n_o) \). Let \( n_1 \) pairs of the form \((r_1 \text{ tuple}, \text{ pointer to } r_2)\) fit in one disk block. Therefore the intermediate result after replacing the \( r_1 \) pointers will occupy \( B_{o2} = \lceil (n_o/n_1) \rceil \) blocks.

2. [14.1] Show that the following equivalences hold. Explain how you can apply them to improve the efficiency of certain queries. *Editing note: I cannot find an outer join symbol in latex, so please read the ugly “\( \Join \)” as “left outer join”.*

(a) \( E_1 \Join \theta (E_2 - E_3) = (E_1 \Join \theta E_2 - E_1 \Join \theta E_3) \).

(b) \( \sigma_\theta (\_A G_F(\mathcal{E})) = \_A G_F(\sigma_\theta(\mathcal{E})) \), where \( \theta \) uses only attributes from \( A \).

(c) \( \sigma_\theta(E_1 \Join E_2) = \sigma_\theta(E_1) \Join E_2 \), where \( \theta \) uses only attributes from \( E_1 \).

**Answer:**
(a) $E_1 \bowtie \theta (E_2 - E_3) = (E_1 \bowtie \theta E_2 - E_1 \bowtie \theta E_3)$.

Let us rename $E_1 \bowtie \theta (E_2 - E_3)$ as $R_1$, $E_1 \bowtie \theta E_2$ as $R_2$, and $E_1 \bowtie \theta E_3$ as $R_3$. It is clear that if a tuple $t$ belongs to $R_1$, it will also belong to $R_2$. If a tuple $t$ belongs to $R_3$, $t[E_3’s$ attributes] will belong to $E_3$, hence $t$ cannot belong to $R_1$. From these two we can say that

$$\forall t, t \in R_1 \Rightarrow t \in (R_1 - R_3).$$

It is clear that if a tuple $t$ belongs to $R_2 - R_3$, then $t[R_2’s$ attributes] $\subseteq E_2$ and $t[R_2’s$ attributes] $\subseteq E_3$. Therefore:

$$\forall t, t \in (R_2 - R_3) \Rightarrow t \in R_1.$$

The above two equations imply the given equivalence.

This equivalence is helpful because evaluation of the right hand side join will produce many tuples which will finally be removed from the result. The left hand side expression can be evaluated more efficiently.

(b) $\sigma_\theta (A_G F(E)) = A_G F(\sigma_\theta(E))$, where $\theta$ uses only attributes from $A$.

$\theta$ uses only attributes from $A$. Therefore if any tuple $t$ in the output of $A_G F(E)$ is filtered out by the selection of the left hand side, all the tuples in $E$ whose value in $A$ is equal to $t[A]$ are filtered out by the selection of the right hand side. Therefore:

$$\forall t, t \notin \sigma_\theta (A_G F(E)) \Rightarrow t \notin A_G F(\sigma_\theta(E)).$$

Using similar reasoning, we can also conclude that

$$\forall t, t \notin A_G F(\sigma_\theta(E)) \Rightarrow t \notin \sigma_\theta (A_G F(E)).$$

The above two equations imply the given equivalence.

This equivalence is helpful because evaluation of the right hand side avoids performing the aggregation on groups which are anyway going to be removed from the result. Thus the right hand side expression can be evaluated more efficiently than the left hand side expression.

(c) $\sigma_\theta (E_1 \bowtie E_2) = \sigma_\theta (E_1) \bowtie E_2$, where $\theta$ uses only attributes from $E_1$.

$\theta$ uses only attributes from $E_1$. Therefore if any tuple $t$ in the output of $(E_1 \bowtie E_2)$ is filtered out by the selection of the left hand side, all the tuples in $E_1$ whose value is equal to $t[E_1]$ are filtered out by the selection of the right hand side. Therefore:

$$\forall t, t \notin \sigma_\theta (E_1 \bowtie E_2) \Rightarrow t \notin \sigma_\theta (E_1) \bowtie E_2.$$  

Using similar reasoning, we can also conclude that

$$\forall t, t \notin \sigma_\theta (E_1) \bowtie E_2 \Rightarrow t \notin \sigma_\theta (E_1 \bowtie E_2).$$

The above two equations imply the given equivalence.

This equivalence is helpful because evaluation of the right hand side avoids producing many output tuples which are anyway going to be removed from the result. Thus the right hand side expression can be evaluated more efficiently than the left hand side expression.
3. [14.4] Consider the relations \( r_1(A, B, C) \), \( r_2(C, D, E) \), and \( r_3(E, F) \), with primary keys \( A \), \( C \), and \( E \), respectively. Assume that \( r_1 \) has 1000 tuples, \( r_2 \) has 1500 tuples, and \( r_3 \) has 750 tuples. Estimate the size of \( r_1 \Join r_2 \Join r_3 \), and give an efficient strategy for computing the join.

**Answer:**

- The relation resulting from the join of \( r_1 \), \( r_2 \), and \( r_3 \) will be the same no matter which way we join them, due to the associative and commutative properties of joins. So we will consider the size based on the strategy of \( ((r_1 \Join r_2) \Join r_3) \). Joining \( r_1 \) with \( r_2 \) will yield a relation of at most 1000 tuples, since \( C \) is a key for \( r_2 \). Likewise, joining that result with \( r_3 \) will yield a relation of at most 1000 tuples because \( E \) is a key for \( r_3 \). Therefore the final relation will have at most 1000 tuples.

- An efficient strategy for computing this join would be to create an index on attribute \( C \) for relation \( r_2 \) and on \( E \) for \( r_3 \). Then for each tuple in \( r_1 \), we do the following:
  
  (a) Use the index for \( r_2 \) to look up at most one tuple which matches the \( C \) value of \( r_1 \).
  
  (b) Use the created index on \( E \) to look up in \( r_3 \) at most one tuple which matches the unique value for \( E \) in \( r_2 \).

4. [14.5] Consider the relations \( r_1(A, B, C) \), \( r_2(C, D, E) \), and \( r_3(E, F) \) of Exercise 14.2. Assume that there are no primary keys, except the entire schema. Let \( V(C, r_1) \) be 900, \( V(C, r_2) \) be 1100, \( V(E, r_2) \) be 50, and \( V(E, r_3) \) be 100. Assume that \( r_1 \) has 1000 tuples, \( r_2 \) has 1500 tuples, and \( r_3 \) has 750 tuples. Estimate the size of \( r_1 \Join r_2 \Join r_3 \), and give an efficient strategy for computing the join.

**Answer:**

The estimated size of the relation can be determined by calculating the average number of tuples which would be joined with each tuple of the second relation. In this case, for each tuple in \( r_1 \), \( 1500/V(C, r_2) = 15/11 \) tuples (on the average) of \( r_2 \) would join with it. The intermediate relation would have \( 15000/11 \) tuples. This relation is joined with \( r_3 \) to yield a result of approximately 10,227 tuples \( (15000/11 \times 750/100 = 10227) \). A good strategy should join \( r_1 \) and \( r_2 \) first, since the intermediate relation is about the same size as \( r_1 \) or \( r_2 \). Then \( r_3 \) is joined to this result.