Problem 1  (Textbook Problem 12.3)

Let relations \( r_1(A,B,C) \) and \( r_2(C,D,E) \) have the following properties: \( r_1 \) has 20,000 tuples, \( r_2 \) has 45,000 tuples, 25 tuples of \( r_1 \) fit on one block, and 30 tuples of \( r_2 \) fit on one block. Estimate the number of block transfers required, using each of the following join strategies for \( r_1 \Join r_2 \):

Answer: \( r_1 \) needs 800 blocks, and \( r_2 \) needs 1500 blocks. Let us assume \( M \) pages of memory. If \( M > 800 \), the join can easily be done in 1500 + 800 disk accesses, using even plain nested-loop join. So we consider on the case where \( M \leq 800 \) pages.

1. Nested-loop join
   Using \( r_1 \) as the outer relation we need 20,000 \( \times \) 1,500 + 800 = 30,000,800 disk accesses, if \( r_2 \) is the outer relation we need 45,000 \( \times \) 800 + 1,500 = 36,001,500 disk accesses.

2. Block nested-loop join
   If \( r_1 \) is the outer relation, \( \lceil \frac{800}{M-2} \rceil \) * 1500 + 800 disk accesses are needed. If \( r_2 \) is the outer relation, \( \lceil \frac{1500}{M-2} \rceil \) * 800 + 1500 disk accesses are needed.

3. Merge join
   Assuming that \( r_1 \) and \( r_2 \) are not initially sorted on the join key, the total sorting cost inclusive of the output is \( B_s = 1500(2\lceil \log_{M-1}1500/M \rceil + 2) + 800(2\lceil \log_{M-1}(800/M) \rceil + 2) \) disk accesses. Assuming all tuples with the same value for the join attributes fit in memory, the total cost is \( B_s + 1500 + 800 \) disk accesses.

4. Hash join
   We assume no overflow occurs. Since \( r_1 \) is smaller, we use it as the build relation and \( r_2 \) as the probe relation. If \( M > 800/M \), i.e. no need for recursive partitioning, then the cost is \( 3(1500 + 800) = 6900 \) disk accesses, else the cost is \( 2(1500 + 800)\lceil \log_{M-1}800 - 1 \rceil + 1500 + 800 \).

Problem 2  (Textbook Problem 13.4)

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:

1. The relation resulting form 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 of \( r_2 \). Likewise, joining that result with \( r_3 \) will yield a relation of at most 1000 tuples because \( E \) is a key of \( r_3 \). Therefore the final relation will have at most 1000 tuples.

2. 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 of \( E \) in \( r_2 \).
Problem 3

Consider the following query on the account(aID, name) and deposit(aID, date, amount) relations:

```
SELECT a.name, d.date, d.amount
FROM deposit AS d
INNER JOIN account AS a
  ON a.aID = d.aID
WHERE amount >= 400
```

Assume that account contains 10,000 accounts, and every account has made 50 deposits on average (the deposit table contains 500,000 deposits total). Both relations are not sorted in any particular order, and there are no indices on the relations. The in-memory buffer can hold up to 12 blocks and there is 100 tuples on average in every block (for both tables). Deposits range from $100 (inclusive) to $500 (exclusive), and you may assume an even distribution.

Answer: The table account has $M = 10,000/100 = 100$ blocks. The table deposit has $N = 500,000/100 = 5,000$ blocks. Let $B$ be the number of blocks in the in-memory buffer, and $B = 12$. The expected number of blocks The expected number of tuples in deposit match the selection predicate is $500,000 * \frac{500 - 400}{500} = 125,000$. The number of blocks needed to store these tuples is $N_s = [125,000/100] = 1,250$.

1. Using a Block Nested-Loop Join, compute the number of block accesses that will be required to perform the operation.

Assume the WHERE clause is evaluated when tuples are joined. Assume the outer relation is account. The number of block accesses is $M + \lceil M/B \rceil * N = 100 + \lceil 500/125 \rceil * 5,000 = 50,100$.

Alternatively, assume WHERE clause is evaluated before tuples are joined. To find $N_s$ blocks it costs $N + N_s$ disk transfers. The total cost is $(N + N_s) + M + \lceil M/B \rceil * N_s = (5,000 + 1,250) + 100 + 10 * 1,250 = 18,850$.

2. Compute the block accesses again using a Merge-Join instead. Both the deposit and account relations remain unordered.

Assume the WHERE clause is evaluated when tuples are joined. The cost of sorting account and deposit is $(2M + 2M * \lceil \log_{256} M/B \rceil) + (2N + 2N * \lceil \log_{256} N/B \rceil) = 400 + 40,000 = 40,400$. The cost of joining the sorted account and deposit is $M + N = 100 + 5,000 = 5,100$. The total cost is $40,400 + 5,100 = 45,500$.

Alternatively, assume WHERE clause is evaluated before tuples are joined. To find $N_s$ blocks it costs $N + N_s$ disk transfers. The total cost is the sum of the costs of selection, merge-sort and join, or $(N + N_s) + (2M + 2M * \lceil \log_{256} M/B \rceil) + (2N + 2N * \lceil \log_{256} N/B \rceil) + (M + N_s) = (5,000 + 1,250) + (2 * 100 + 2 * 100 * \lceil \log_{128} 100/128 \rceil) + (2 * 1,250 + 2 * 1,250 * \lceil \log_{128} 250/128 \rceil) + (100 + 1,250) = 15,500$.

3. Now let’s add some indices to these tables. First let’s add a primary index on deposit.amount. The account relation remains unordered. Again, using a Block Nested-Loop Join, compute the number of block accesses that will be required to perform the operation. Assume that the WHERE clause is evaluated before the JOIN, with an index fan out of 100.

The data entry in the tree index contains a key and an actual tuple. The height of the tree is $H = \lceil \log_{100} 500,000/(num.entries/leaf) \rceil = \lceil \log_{100} 500,000/100 \rceil = 2$. To evaluate WHERE clause, all $N_s$ blocks in deposit can be retrieved through a leaf-scan. The cost is $H = H + N_s = 1,252$.

The WHERE clause is evaluated before tuples are joined. The first way is to use account as the outer relation. We need to materialize $N_s$ blocks first. To find $N_s$ blocks it costs $N_i + N_s$ disk transfers. The total cost is $(N_i + N_s) + (M + \lceil M/(B-2) \rceil * N_s) = (1,252 + 1,250) + (100 + \lceil 100/(12-2) \rceil * 1,250) = 15,102$.

An alternative way is to use deposit as the outer relation, which does not require materialization. The total cost is $N_i + \lceil N_s/(B-2) \rceil * M = 1,252 + \lceil 1,250/(12-2) \rceil * 100 = 13,752$.

(Note: If the WHERE clause is evaluated when tuples are joined, the total cost is $M + \lceil M/(B-2) \rceil * N_i = 100 + \lceil 100/(12-2) \rceil * 1,252 = 12,620$. This is not the assumption of this problem, though.)
4. Assume that the primary index is now changed to deposit.aID, rather than deposit.amount. The account relation remains unordered. Using a Indexed Nested-Loop Join, compute the number of block accesses that will be required to perform the operation.

With the same tree index format assumption, the tree height is the same as above, $H = 2$. For deposit, the expected cost of looking up a tuple through the tree index is $C_s = H + 1 = 3$. Assume account is the outer relation. The expected total cost of the join is $M + |\text{account}| * C_s = 100 + 10,000 * 3 = 30,100$.

5. A primary index is added on account.aID. Now that both relations are sorted, recompute the number of block accesses using a Merge-Join.

Assume the WHERE clause is evaluated when tuples are joined. Because we have primary indices on both tables, a merge join costs $M + N = 5,100$.

Alternatively, assume WHERE clause is evaluated before tuples are joined. To find $N_s$ blocks it costs $\log_2 N + N_s$ disk transfers. The total cost is $(\log_2 N + N_s) + M = ([\log_2 5,000] + 1,250) + 100 = 1,363$.

Problem 4

Consider a database with the following initial values, and the attached command log:

$A = 50, B = 48, C = 0, D = 47$

LOG:

< T₀, start >
< T₀, A, 50, 75 >
< T₁, start >
< T₁, B, 48, 92 >
< T₂, start >
< T₂, C, 0, 33 >
< T₁, B, 92, 108 >
< checkpoint : T₀, T₁, T₂ >
< T₃, start >
< T₀, A, 75, 100 >
< T₂, commit >
< T₃, D, 47, 52 >
< T₃, commit >

Assume that the system crashes before the remaining transactions can commit. Use the recovery protocol for concurrent transactions (which persists all in-memory dirty pages and transaction log entries at each checkpoint) to answer the following questions.

1. List any transactions that will need to be undone or redone in the recovery process.

   Answer: $T₀$ and $T₁$ will need to be undone. $T₂$ and $T₃$ will need to be redone.

2. List, in order, the set of logged operations to be performed to undo or redo the transactions. (i.e. "Set A to 7", "Set B to 39", etc.)

   Answer:

   // UNDO
   Set B to 92;
   Set B to 48;
   Set A to 50;
   // REDO
   Set D to 52;
3. Give the final values for \( A, B, C, \) and \( D. \)

Answer: \( A = 50, B = 48, C = 33, \) and \( D = 52. \)

**Problem 5**

Assume the following transactions: \( T_1, T_2 \) and \( T_3: \)

<table>
<thead>
<tr>
<th>Schedule</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
</tbody>
</table>

Answer the following questions for the schedule:

1. Is the schedule serializable? Justify your answer by drawing the precedence graph.

   Answer: Yes, the schedule is serializable. Please find the acyclic precedence graph below\(^1\).

   ![Precedence Graph](image)

2. Assume that all three transactions begin and end at the same time. Could the schedule be produced by the two-phase locking protocol? Insert lock and unlock operations into the schedule to justify your answer.

   Answer: Yes.

\(^1\)Note that the numeric values of time do not carry any significance beyond discretizing the time. One can map different values as long as the result amounts to the same discretization.
3. Making the same assumptions, could the schedule be produced by the strict two-phase locking protocol? Insert lock and unlock operations into the schedule to justify your answer.

Answer: No, strict 2PL cannot produce this schedule given the assumption that T1, T2 and T3 start and end at the same time. Suppose otherwise. By strict 2PL, T2 read(C) needs to happen after T3 ends, because T3 must release the exclusive lock on C after it has ended. But this implication contradicts the assumption that T2 and T3 end at the same time.