Warmup #1  (Textbook Problem 13.15)

Suppose that a B+-tree index on (dept_name, building) is available on relation department. What would be the best way to handle the following selection?

\[ \sigma(\text{building}<"\text{Watson}" \land \text{budget}<55000 \land \text{dept_name}="\text{Music}")(\text{department}). \]

Warmup #2  (Textbook Problem 14.7)

What is a cascadeless schedule? Why is cascadelessness of schedules desirable? Are there any circumstances under which it would be desirable to allow noncascadeless schedules? Explain your answer.

Warmup #3  (Textbook Problem 14.15)

Consider the following two transactions:

\[ T_{13}: \text{ read}(A); \]
\[ \text{ read}(B); \]
\[ \quad \text{ if } A = 0 \text{ then } B := B + 1; \]
\[ \text{ write}(B). \]

\[ T_{14}: \text{ read}(B); \]
\[ \text{ read}(A); \]
\[ \quad \text{ if } B = 0 \text{ then } A := A + 1; \]
\[ \text{ write}(A). \]

Let the consistency requirement be \( A = 0 \lor B = 0 \), with \( A = B = 0 \) the initial values.

1. Show that every serial execution involving these two transactions preserves the consistency of the database.
2. Show a concurrent execution of \( T_{13} \) and \( T_{14} \) that produces a nonserializable schedule.
3. Is there a concurrent execution of \( T_{13} \) and \( T_{14} \) that produces a serializable schedule?

Problem 4 (To Be Graded)

Consider the following query for the \( \text{emp}(id, dept, salary) \) relation:

\[
\text{SELECT } \text{dept, avg(salary)} \\
\text{FROM emp} \\
\text{GROUP BY dept}
\]
There are two major strategies in most DBMSes for computing aggregates. In terms of PostgreSQL they are called: GroupAggregate and HashAggregate.

- GroupAggregate assumes \( emp \) is sorted on \( dept \). If not, it performs the sorting first. Then it just scans it and outputs groups one by one.
- HashAggregate, performs hashing with the function \( h(emp) = dept \), computes the aggregate for each group and then outputs the result.

Assume that \( emp \) contains 20,000 employees, every department contains 2,000 employees on average. The relation is not sorted in any particular order. The in-memory buffer can hold up to 10 blocks and there is 100 tuples on average in every block.

1. Compute the cost of computing both the first result and all of the results in terms of disk seeks and block reads. For HashAggregate, assume the memory is enough to hold all individual groups (this is fair, since you have to store only (sum, count) pairs for every department). For sorting purposes assume external merge-sort.

2. Assume the ORDER BY dept clause is added to the query above. How do the costs change?

3. Assume the initial relation is sorted by dept on disk. What about the costs now?

**Problem 5 (To Be Graded)**

Assume the following transactions: \( T1, T2 \) and \( T3 \):

<table>
<thead>
<tr>
<th>Schedule</th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td></td>
<td>read(A)</td>
</tr>
<tr>
<td>2</td>
<td>read(A)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>-</td>
<td>write(C)</td>
</tr>
<tr>
<td>4</td>
<td>-</td>
<td>read(B)</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>read(C)</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>write(B)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Answer the following questions for the schedule:

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

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.

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.