Warmup #1 (Textbook Problem 13.15)

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

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

Answer: Use B+-tree to look up the first leaf node that may contain the key \((\text{dept\_name} = \text{"Music"}, \text{building} = \text{"Watson"})\). Starting from this leaf node, scan subsequent leaves by iterating over all keys until \(\text{dept\_name} < \text{"Music"}\). For each key, retrieve the corresponding tuple and check against the predicate \(\land (\text{budget} < 55000)\), and output the tuples that match the predicate.

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.

Answer: A cascadeless schedule is one where, for each pair of transactions \(T_i\) and \(T_j\) such that \(T_j\) reads data items previously written by \(T_i\), the commit operation of \(T_i\) appears before the read operation of \(T_j\). Cascadeless schedules are desirable because the failure of a transaction does not lead to the aborting of any other transaction. Of course this comes at the cost of less concurrency. If failures occur rarely, so that we can pay the price of cascading aborts for the increased concurrency, noncascadeless schedules might be desirable.

Warmup #3 (Textbook Problem 14.12)

List the ACID properties. Explain the usefulness of each.

Answer:

1. Atomicity. Either all operations of the transaction are reflected properly in the database, or none are.

2. Consistency. Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

3. Isolation. Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions \(T_i\) and \(T_j\), it appears to \(T_i\) that either \(T_j\) finished execution before \(T_i\) started or \(T_j\) started execution after \(T_i\) finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.

4. Durability. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.
Warmup #4  (Textbook Problem 14.15)

Consider the following two transactions:

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

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

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.

   Answer: There are two possible executions: \( T_{13}T_{14} \) and \( T_{14}T_{13} \).

   \[
   \begin{array}{c|cc}
   \text{Case 1:} & A & B \\
   \text{initially} & 0 & 0 \\
   \text{after } T_{13} & 0 & 1 \\
   \text{after } T_{14} & 0 & 1 \\
   \text{Consistency met: } & A = 0 \lor B = 0 \equiv T \lor F \equiv T \\
   \end{array}
   \]

   \[
   \begin{array}{c|cc}
   \text{Case 2:} & A & B \\
   \text{initially} & 0 & 0 \\
   \text{after } T_{14} & 1 & 0 \\
   \text{after } T_{13} & 1 & 0 \\
   \text{Consistency met: } & A = 0 \lor B = 0 \equiv F \lor T \equiv T \\
   \end{array}
   \]

2. Show a concurrent execution of \( T_{13} \) and \( T_{14} \) that produces a nonserializable schedule.

   Answer: Any interleaving of \( T_{13} \) and \( T_{14} \) results in a non-serializable schedule. For example,

   \[
   T_{13} \quad T_{14}
   \]

   read(A)  
   read(B)  
   read(A)  
   if B = 0 then A := A + 1  
   write(A)  

   read(B)  
   if A = 0 then B := B + 1  
   write(B)  

3. Is there a concurrent execution of \( T_{13} \) and \( T_{14} \) that produces a serializable schedule?

   No, there is no concurrent execution of \( T_{13} \) and \( T_{14} \) that produces a serializable schedule. A serializable schedule results in \( A = 0 \lor B = 0 \). Suppose we start with \( T_{13} \text{read}(A) \). When the schedule ends, no matter when we run steps of \( T_{14} \), \( B = 1 \). Now suppose we start executing \( T_{14} \) prior to the completion of \( T_{13} \). Then \( T_{14} \text{read}(B) \) will give \( B \) a value of 0. When \( T_{14} \) completes, \( A = 1 \). Thus \( A = B = 1 \) violates the consistency requirement. The argument is similar for starting with \( T_{14} \text{read}(B) \).
Problem 5 (To Be Graded)

A database stores warehouses and products. Here is the schema:

```sql
create table Warehouse(
    name varchar(100) primary key,
    zip_code varchar(20) not null,
    index(zip_code));

create table Product(
    id int primary key,
    description varchar(600) not null,
    availableAt varchar(100),
    index(availableAt),
    foreign key(availableAt) references Warehouse(name)
);```

There are 500,000 products and 10,000 warehouses in the database. The average size of a warehouse record is 50 bytes and the average size of a product record is 100 bytes. All indexes are B+-trees. There are 500 known zip codes. You want to find all products stored in Providence warehouses at the zip code "02904-2413".

With the assumptions that:

- An index lookup takes approximately 20ms.
- Scanning through a table takes 0.1 ms per block.
- The disk block size is 8K = 8192 bytes.
- The buffer size is 4 blocks.

For each query plan, find the approximate runtime; discuss which plan should be selected and why.

Consider the following query plans:

1. Select Providence warehouses (02904-2413), and then join with product using an index nested loop join. Answer:

   **selection cost:**
   - Assume there is 1 area code for PVD
   - There are 10,000/500 = 20 warehouses in Providence
   - 20*50 = 1000 bytes, so only one block

   **index lookup + scan = 20 + 20 * 0.1 ms = 22 ms**

   For index nested loop join R is warehouse selection since Product still has an index:
   - B_r = 1 blocks
   - n_r = 20 tuples
   - c = 20 ms (cost to read index of availableAt for products)
   - 1*0.1 (cost to read block) + 20*(20 ms)
   - = 402 ms

   **total time = 402 + 22 = ~424 ms**

2. Select Providence warehouses (02904-2413), and then join with Product using a sort-merge join.
blocks in products
B_r = 6173 blocks because 81 product records per block and # of blocks is (500,000/81)
blocks in warehouses
B_s = 1 block
cost to sort is:
2* block_size (log_base(m-1) (block_size / M) + 1)
cost for sort on products:
= 2 * 6173 * (7 + 1) = 98,768
cost for sort on warehouses:
remember buffer is bigger than we need so we can sort all in memory
= 2 * 1 * (1 + 1) = 4
total cost = 0.1 * (6173 + 1 + 98,768 + 4) = 10,495 ms

3. Join Product with Warehouse using an index nested loop and then select the merchandise in (02904-2413).

For index nested loop join R is warehouse since it is still a smaller relation:
B_r = (10,000*50)/8192 = 62
n_r = 10,000
c = 20 ms
62*0.1 + 10,000*20
= 200,006.2 ms

Selection cost:

Using the index on zipcode:
20 + 1000 * 0.1 = 120 ms

'Scan the whole relation’ approach:
(500,000 * 150 bytes / 8192 bytes) * 0.1 ms = 915.5 ms

total cost = join cost + selection cost > 200,000 ms

Answer:
Originally for this problem the right answer was to not use the first plan since it’s not possible to run an index nested loop join, since the availableAt column on Product is not indexed. Without the index, the first plan will be much more expensive than the second. Therefore the second plan is the optimal one.

Considering the changes made to the problem, the first plan will perform the best; therefore the plan should be: Select Providence warehouses (02904-2413), and then join with product using an index nested loop join.
Problem 6 (To Be Graded)

Consider the following transactions. Operations are to be executed in order from top to bottom.

<table>
<thead>
<tr>
<th>Schedule A</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>write(H)</td>
</tr>
<tr>
<td>read(G)</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>write(B)</td>
</tr>
<tr>
<td>write(C)</td>
</tr>
<tr>
<td>write(E)</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>-</td>
</tr>
<tr>
<td>-</td>
</tr>
</tbody>
</table>

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.

T1 \rightarrow T4 \rightarrow T5 \rightarrow T2 \rightarrow T3 \rightarrow T6

\(^1\)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.

\(^2\)Note that there are multiple acceptable solutions here