Quiz 3 Study Guide
Join Cost Estimation
Cardinality of Joins in general

Assume join: \( R \Join S \) (common attributes are not keys)

1. If \( R, S \) have no common attributes: \( n_r \times n_s \)
2. If \( R, S \) have attribute \( A \) in common:

\[
\frac{n_r}{V(A,s)} \text{ or } \frac{n_s}{V(A,r)} \tag{take min}
\]

- These are not the same when \( V(A,s) \neq V(A,r) \).
- When this is true, there are likely to be dangling tuples.
- Thus, the smaller is likely to be more accurate.
Know how to compute costs of different join strategies

• Nested Loop Join
• Block Nested Loop Join
• Indexed Nested Loop Join
• Merge Join
• Sort-Merge Join (requires you to know the cost of sorting a relation)
• Hash Join

Be wary of special cases; what is the cost when one relation fits entirely in memory?
Transaction Processing
ACID Properties

Properties that a Xaction needs to have:

- **Atomicity**: either all operations in a Xaction take effect, or none

- **Consistency**: operations, taken together preserve db consistency

- **Isolation**: intermediate, inconsistent states must be concealed from other Xactions

- **Durability**: If a Xaction successfully completes (“commits”), changes made to db must persist, even if system crashes
• Serializability
• Conflict Equivalency
• Conflict Serializability
• Precedence Graphs
• Cascading Rollback
• Cascade-less Schedules
Concurrency Control
• 2PL

• Strict 2PL

• Deadlock Detection

• Recovering from a deadlock
Idea: If action $p_i$ of Xact $T_i$ conflicts with action $q_j$ of Xact $T_j$, and $TS(T_i) < TS(T_j)$, then $p_i$ must occur before $q_j$. Otherwise, restart violating Xact.
When Xact T wants to read Object O

- If $TS(T) < W-TS(O)$, this violates timestamp order of T w.r.t. writer of O.
  - So, abort T and restart it with a new, larger TS. (If restarted with same TS, T will fail again!)

- If $TS(T) > W-TS(O)$:
  - Allow T to read O.
  - Reset R-TS(O) to max(R-TS(O), TS(T))

- Change to R-TS(O) on reads must be written to disk! This and restarts represent overhead.
When Xact T wants to Write Object O

- If $TS(T) < R-TS(O)$, then the value of $O$ that $T$ is producing was needed previously, and the system assumed that that value would never be produced. **write rejected**, $T$ is rolled back.

- If $TS(T) < W-TS(O)$, then $T$ is attempting to write an obsolete value of $O$. Hence, this **write operation is rejected**, and $T$ is rolled back.

- Otherwise, the **write operation is executed**, and $W-TS(O)$ is set to $TS(T)$. 

---

**Diagram:**

- U reads O
- T writes O
- T start
- U start
Problem 1

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>

Assume that the transactions begin at the following timestamps,

$T_1 : 2, T_2 : 4, T_3 : 1$.

All transactions will commit at the same time. Will any transactions in this schedule abort if we are using the time-stamp protocol? Which one(s)?
Recovery
Immediate Database Modification (Cont.)

- Recovery procedure:
  - **Undo**: \(<T_i, \text{start}>\) is in the log but \(<T_i, \text{commit}>\) is not. **Undo**:
    - restore the value of all data items updated by \(T_i\) to their old values, going backwards from the last log record for \(T_i\)
  - **Redo**: \(<T_i, \text{start}>\) and \(<T_i, \text{commit}>\) are both in the log.
    - sets the value of all data items updated by \(T_i\) to the new values, going forward from the first log record for \(T_i\)

- Both operations must be **idempotent**: even if the operation is executed multiple times the effect is the same as if it is executed once
Checkpoints (Cont.)

Recovering from log with checkpoints:

1. Scan backwards from end of log to find the most recent `<checkpoint>` record

2. Continue scanning backwards till a record `<T_i start>` is found.

3. Need only consider the part of log following above `start` record. Why?

4. After that, recover from log with the rules that we had before.
Problem 2

Consider a database with the following initial values, and the attached command log:
\[ A = 50, B = 48, C = 0, D = 47 \]

LOG:
\[
\begin{align*}
<T_0, \text{start}> \\
<T_0, A, 50, 75> \\
<T_1, \text{start}> \\
<T_1, B, 48, 92> \\
<T_2, \text{start}> \\
<T_2, C, 0, 33> \\
<T_1, B, 92, 108> \\
<\text{checkpoint} : T_0, T_1, T_2> \\
<T_3, \text{start}> \\
<T_0, A, 75, 100> \\
<T_2, \text{commit}> \\
<T_3, D, 47, 52> \\
<T_3, \text{commit}>
\end{align*}
\]

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.

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.)

3. Give the final values for A, B, C, and D.
Good luck! :}