QUIZ 3
REVIEW SESSION
DATABASE MANAGEMENT SYSTEMS
Subject Areas

1. Query Processing
2. Concurrency Control
3. Recovery
Query Processing

1. Know common join and selection algorithms and costs
   a. Linear Search
   b. Binary Search
   c. Index Scan
   d. Basic Join
   e. Nested Loop (you already know this one!)
   f. Block Nested Loop (you already know this one!)
   g. Merge Join
   h. External Sorting
   i. Hash Join
Merge Join

Only use for equality
Two indexes sorted on A
Cost: $b_r + b_s$
If need to sort $R,S$: $b_r + b_s + \text{sort}(r) + \text{sort}(s)$
External Sorting

Method of sorting data onto blocks
- Want to minimize block accesses (rather than say CPU cycles)

Merge Sort best sort algorithm

Cost: $2 b_R \cdot (\text{ciel}(\log M - 1 (b_R / M)) + 1)$

$M$ is size of buffer (in blocks)

$\text{ciel}(\log M - 1 (b_R / M))$ is number of iterations
Hash Join

Equality join on attribute; apply hash to both relations

Cost: \(3(b_r + b_s) + 4 \ n_h\)

\(n_h = \text{ciel}(b_s / M) \times f\)

Normally function chosen so that \(s_1\) hold in memory

\(f \approx 1.2\) typically

\(M = \text{size of buffer (in blocks)}\)

If no partitioning needed then just \(b_r + b_s\)
Probably Good Idea to Write Down (or copy)

1. Size (# tuples in r) \( n_r \)
2. Size (# blocks in r) \( b_r \)
3. Block size (# tuples per block) \( f_r \)
   (typically \( b_r = \left\lfloor \frac{n_r}{f_r} \right\rfloor \))
4. Tuple size (in bytes) \( s_r \)
5. Attribute Values \( V(\text{att}, r) \)
   (for each attribute \text{att} in r, # of different values)
6. Selection Cardinality \( SC(\text{att}, r) \)
   (for each attribute \text{att} in r,
    expected size of a selection: \( \sigma_{\text{att} = K}(r) \))
Concurrency Management

1. ACID
2. Levels of Serializability
3. Cascading Rollbacks and Recoverability
4. Two Phase Locking Protocol
5. Deadlock
   a. Wait and die
6. Timestamp based protocols
ACID

• **Atomicity**: All actions in the Xaction happen, or none happen.

• **Consistency**: If each Xaction is consistent, and the DB starts consistent, it ends up consistent.

• **Isolation**: Execution of one Xaction is isolated from that of other Xacts.

• **Durability**: If a Xaction commits, its effects persist.
Serializability

1. Serializable - effect on DB the same
2. Conflict Serializable - conflict equivalent to a serial schedule
   i. Use Precedence graph to determine
3. View Serializable - SQL reads/writes same values
2PL

- This is a protocol which ensures conflict-serializable schedules.
- Phase 1: Growing Phase
  - transaction may obtain locks
  - transaction may not release locks
- Phase 2: Shrinking Phase
  - transaction may release locks
  - transaction may not obtain locks
- The protocol ensures serializability.
Deadlock

- **wait-die scheme — non-preemptive**
  - older transaction may wait for younger one to release data item.
  - younger transactions never wait for older ones; they are rolled back instead.

- **wound-wait scheme — preemptive**
  - older transaction wounds (forces rollback) of younger transaction instead of waiting for it.
  - younger transactions may wait for older ones.

<table>
<thead>
<tr>
<th></th>
<th>Wait-Die</th>
<th>Wound-Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>O needs a resource</td>
<td>O Waits</td>
<td>Y Dies</td>
</tr>
<tr>
<td>held by Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y needs a resource</td>
<td>Y Dies</td>
<td>Y Waits</td>
</tr>
<tr>
<td>held by O</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Timestamp Protocol

If action pi of Xact Ti conflicts with action qj of Xact Tj, and TS(Ti) < TS(Tj), then pi must occur before qj. Otherwise, restart violating Xact.

- 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).

Thomas Write Rule: if write is obsolete (further timestamp already wrote, and no reads) then do not have to roll back
Recovery

1. ACID
   a. How does recovery apply here?

2. Logging
   a. Immediate vs. Deferred Modification
   b. Checkpoints (See simpleDB!)

3. Shadow Paging
Logging

Immediate:
Undo: if <Ti, start> is in the log but <Ti commit> is not.
How: restore the value of all data items updated by Ti to their old values, going backwards from the last log record for Ti

Redo: if <Ti start> and <Ti commit> are both in the log.
How: sets the value of all data items updated by Ti to the new values, going forward from the first log record for Ti

Deferred:
Redo: if both <Ti start> and <Ti commit> are there in the log

All logging operations should be idempotent
Shadow Paging

Maintain clean second copy of data pages; only switch after transaction commits

Advantages
- No overhead of writing log records
- Recovery is trivial

Disadvantages:
- Copying the entire page table is very expensive
- Data gets fragmented
- Hard to extend for concurrent transactions
Good luck!

Remember you may bring a double sided 9.5” x 11” sheet of notes (write down those QP equations!)

Quiz is Wednesday in Barus and Holley 166