Lecture 9-10: Recovery

Friday, April 16 and Monday, April 19, 2010
Outline

- Disks 13.2
- Undo logging 17.2
- Redo logging 17.3
- Redo/undo 17.4
Project 2

What you will learn:
• Connect to db and call SQL from java (read 9.6)
• Dependent joins
• Integrate two databases
• Transactions

Amount of work:
• 20 SQL queries + 180 lines Java ≈ 12 hours (?)
Project 2

• Database 1 = IMDB on SQL Server

• Database 2 = you create a CUSTOMER db on postgres
  – Customers
  – Rentals
  – Plans
The Mechanics of Disk

Mechanical characteristics:
• Rotation speed (5400RPM)
• Number of platters (1-30)
• Number of tracks (<=10000)
• Number of bytes/track($10^5$)

Unit of read or write: disk block
Once in memory: page
Typically: 4k or 8k or 16k
RAID

Several disks that work in parallel
- Redundancy: use parity to recover from disk failure
- Speed: read from several disks at once

Various configurations (called *levels*):
- RAID 1 = mirror
- RAID 4 = n disks + 1 parity disk
- RAID 5 = n+1 disks, assign parity blocks round robin
- RAID 6 = “Hamming codes”

Not required for exam, but interesting reading in the book
Disk Access Characteristics

- **Disk latency** = time between when command is issued and when data is in memory

- **Disk latency** = seek time + rotational latency
  - Seek time = time for the head to reach cylinder
    - 10ms – 40ms
  - Rotational latency = time for the sector to rotate
    - Rotation time = 10ms
    - Average latency = 10ms/2

- Transfer time = typically 40MB/s
- Disks read/write one block at a time

Large gap between disk I/O and memory ➔ Buffer pool
Buffer Management in a DBMS

Page Requests from Higher Levels

• Data must be in RAM for DBMS to operate on it!
• Table of <frame#, pageid> pairs is maintained

choice of frame dictated by replacement policy

Dan Suciu -- 444 Spring 2010
Buffer Manager

Page replacement policies

• LRU = expensive
• Clock algorithm = cheaper alternative

Both work well in OS, but not always in DB
Least Recently Used (LRU)

1. Initial state: P5, P2, P8, P4, P1, P9, P6, P3, P7
2. Read(P6): P6, P5, P2, P8, P4, P1, P9, P3, P7
3. Read(P10): P10, P6, P5, P2, P8, P4, P1, P9, P3
4. Input(P10): P10, P6, P5, P2, P8, P4, P1, P9, P3
Buffer Manager

DBMS build their own buffer manager and don’t rely on the OS

• Better control for transactions
  – Force pages to disk
  – Pin pages in the buffer

• Tweaks to LRU/clock algorithms for specialized accesses, s.a. sequential scan
Transaction Management and the Buffer Manager

The transaction manager operates on the buffer pool

• **Recovery**: ‘log-file write-ahead’, then careful policy about which pages to force to disk

• **Concurrency control**: locks at the page level, multiversion concurrency control
Transaction Management

Two parts:

• Recovery from crashes: **ACID**
• Concurrency control: **ACID**

Both operate on the buffer pool
## Recovery

<table>
<thead>
<tr>
<th>Type of Crash</th>
<th>Prevention</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrong data entry</td>
<td>Constraints and Data cleaning</td>
</tr>
<tr>
<td>Disk crashes</td>
<td>Redundancy: e.g. RAID, archive</td>
</tr>
<tr>
<td>Fire, theft, bankruptcy…</td>
<td>Remote backups</td>
</tr>
<tr>
<td>System failures: e.g. power</td>
<td>DATABASE RECOVERY</td>
</tr>
</tbody>
</table>
Main Idea for Recovery

• Write-ahead log =
  – A file that records every single action of all running transactions

  – After a crash, transaction manager reads the log and finds out exactly what the transactions did or did not
Transactions

• Assumption: the database is composed of elements
  – Usually 1 element = 1 block
  – Can be smaller (=1 record) or larger (=1 relation)

• Assumption: each transaction reads/writes some elements
Primitive Operations of Transactions

- **READ(X,t)**
  - copy element X to transaction local variable t
- **WRITE(X,t)**
  - copy transaction local variable t to element X
- **INPUT(X)**
  - read element X to memory buffer
- **OUTPUT(X)**
  - write element X to disk
Example

START TRANSACTION
READ(A,t);
t := t*2;
WRITE(A,t);
READ(B,t);
t := t*2;
WRITE(B,t)
COMMIT;

Atomicity:
BOTH A and B are multiplied by 2
REDA(A,t); t := t*2; WRITE(A,t);
REDA(B,t); t := t*2; WRITE(B,t)

<table>
<thead>
<tr>
<th>Action</th>
<th>t</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT(A)</td>
<td></td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>Action</td>
<td>t</td>
<td>Mem A</td>
<td>Mem B</td>
<td>Disk A</td>
<td>Disk B</td>
</tr>
<tr>
<td>------------</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>INPUT(A)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>

Crash occurs after OUTPUT(A), before OUTPUT(B)
We lose atomicity
The Log

- An append-only file containing log records
- Multiple transactions run concurrently, log records are interleaved
- After a system crash, use log to:
  - Redo some transaction that didn’t commit
  - Undo other transactions that didn’t commit
- Three kinds of logs: undo, redo, undo/redo
Undo Logging

Log records

• <START T>
  – transaction T has begun

• <COMMIT T>
  – T has committed

• <ABORT T>
  – T has aborted

• <T,X,v>
  – T has updated element X, and its old value was v
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT(A)</td>
<td></td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8&gt;</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,8&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
<tr>
<td>Action</td>
<td>T</td>
<td>Mem A</td>
<td>Mem B</td>
<td>Disk A</td>
<td>Disk B</td>
<td>Log</td>
</tr>
<tr>
<td>------------</td>
<td>---</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>INPUT(A)</td>
<td></td>
<td></td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8&gt;</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,8&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
</tbody>
</table>

WHAT DO WE DO?
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT(A)</td>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
<td>&lt;T,A,8&gt;</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
<td></td>
<td>&lt;T,B,8&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
</tbody>
</table>

WHAT DO WE DO ?

Crash !
After Crash

• In the first example:
  – We UNDO both changes: A=8, B=8
  – The transaction is atomic, since none of its actions has been executed

• In the second example
  – We don’t undo anything
  – The transaction is atomic, since both it’s actions have been executed
Undo-Logging Rules

U1: If T modifies X, then <T,X,v> must be written to disk before OUTPUT(X)

U2: If T commits, then OUTPUT(X) must be written to disk before <COMMIT T>

• Hence: OUTPUTs are done *early*, before the transaction commits
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;START T&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INPUT(A)</td>
<td></td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8&gt;</td>
</tr>
<tr>
<td>INPUT(B)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>16</td>
<td>8</td>
<td>&lt;T,B,8&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>COMMIT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
</tbody>
</table>

The table shows the actions, time, and storage locations for the given transactions. The log entries are updated accordingly.
Recovery with Undo Log

After system’s crash, run recovery manager

• Idea 1. Decide for each transaction T whether it is completed or not
  – <START T>….<COMMIT T>…. = yes
  – <START T>….<ABORT T>……… = yes
  – <START T>…………………… = no

• Idea 2. Undo all modifications by incomplete transactions
Recovery with Undo Log

Recovery manager:

- Read log from the end; cases:
  - <COMMIT T>: mark T as completed
  - <ABORT T>: mark T as completed
  - <T,X,v>: if T is not completed
    then write X=v to disk
    else ignore
  - <START T>: ignore
Recovery with Undo Log

Question 1: Which updates are undone?

Question 2: What happens if there is a second crash, during recovery?

Question 3: How far back do we need to read in the log?
Recovery with Undo Log

• Note: all undo commands are idempotent
  – If we perform them a second time, no harm is done
  – E.g. if there is a system crash during recovery, simply restart recovery from scratch
Recovery with Undo Log

When do we stop reading the log?

• We cannot stop until we reach the beginning of the log file
• This is impractical

Instead: use checkpointing
Checkpointing

Checkpoint the database periodically

• Stop accepting new transactions
• Wait until all current transactions complete
• Flush log to disk
• Write a <CKPT> log record, flush
• Resume transactions
Undo Recovery with Checkpointing

During recovery, Can stop at first <CKPT>

other transactions

transactions T2,T3,T4,T5

other transactions
Nonquiescent Checkpointing

- Problem with checkpointing: database freezes during checkpoint
- Would like to checkpoint while database is operational
- Idea: nonquiescent checkpointing

Quiescent = being quiet, still, or at rest; inactive
Non-quiescent = allowing transactions to be active
Nonquiescent Checkpointing

• Write a \(<\text{START CKPT}(T_1, \ldots, T_k)\)>
  where $T_1, \ldots, T_k$ are all active transactions
• Continue normal operation
• When all of $T_1, \ldots, T_k$ have completed, write
  \(<\text{END CKPT}\)>
Undo Recovery with Nonquiescent Checkpointing

During recovery,
Can stop at first
<CKPT>

Q: do we need
<END CKPT> ?

... ... ...
... ...
<START CKPT T4, T5, T6>
...
...
...
<END CKPT>
...
...
...

earlier transactions plus
T4, T5, T6

T4, T5, T6, plus
later transactions

later transactions
Implementing ROLLBACK

• A transaction ends in COMMIT or ROLLBACK
• Use the undo-log to implement ROLLBACK

• LSN = Log Sequence Number
• Log entries for the same transaction are linked, using the LSN’s
• Read log in reverse, using LSN pointers
Redo Logging

Log records

• \(<\text{START } T> = \text{ transaction } T \text{ has begun}\)
• \(<\text{COMMIT } T> = T \text{ has committed}\)
• \(<\text{ABORT } T> = T \text{ has aborted}\)
• \(<T,X,v> = T \text{ has updated element } X, \text{ and its new value is } v\)
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,16&gt;</td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,16&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&lt;COMMIT T&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Redo-Logging Rules

R1: If T modifies X, then both \(<T,X,v>\) and \(<\text{COMMIT T}>\) must be written to disk before OUTPUT(X)

• Hence: OUTPUTs are done *late*
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,16&gt;</td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,16&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
</tbody>
</table>
Recovery with Redo Log

After system’s crash, run recovery manager

• Step 1. Decide for each transaction T whether we need to redo or not
  – <START T>….<COMMIT T>…. = yes
  – <START T>….<ABORT T>…….. = no
  – <START T>……………………. = no

• Step 2. Read log from the beginning, redo all updates of committed transactions
Recovery with Redo Log

\[
\langle \text{START T1} \rangle
\langle T1, X1, v1 \rangle
\langle \text{START T2} \rangle
\langle T2, X2, v2 \rangle
\langle \text{START T3} \rangle
\langle T1, X3, v3 \rangle
\langle \text{COMMIT T2} \rangle
\langle T3, X4, v4 \rangle
\langle T1, X5, v5 \rangle
\ldots
\ldots
\]

Dan Suciu -- 444 Spring 2010
Nonquiescent Checkpointing

• Write a `<START CKPT(T₁,…,Tk)>` where T₁,…,Tk are all active transactions
• Flush to disk all blocks of committed transactions (*dirty blocks*), while continuing normal operation
• When all blocks have been flushed, write `<END CKPT>`
Redo Recovery with Nonquiescent Checkpointing

Step 1: look for
The last
<END CKPT>

All OUTPUTs of T1 are guaranteed to be on disk

Cannot use

Step 2: redo from the earliest start of T4, T5, T6 ignoring transactions committed earlier

All OUTPUTs of T1 are guaranteed to be on disk
Comparison Undo/Redo

• Undo logging:
  – OUTPUT must be done early
  – If <COMMIT T> is seen, T definitely has written all its data to disk (hence, don’t need to redo) – inefficient

• Redo logging
  – OUTPUT must be done late
  – If <COMMIT T> is not seen, T definitely has not written any of its data to disk (hence there is not dirty data on disk, no need to undo) – inflexible

• Would like more flexibility on when to OUTPUT: undo/redo logging (next)
Undo/Redo Logging

Log records, only one change

- \(<T,X,u,v>\) = \(T\) has updated element \(X\), its \textit{old} value was \(u\), and its \textit{new} value is \(v\)
Undo/Redo-Logging Rule

UR1: If T modifies X, then <T,X,u,v> must be written to disk before OUTPUT(X)

Note: we are free to OUTPUT early or late relative to <COMMIT T>
<table>
<thead>
<tr>
<th>Action</th>
<th>T</th>
<th>Mem A</th>
<th>Mem B</th>
<th>Disk A</th>
<th>Disk B</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAT(A,t)</td>
<td>8</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td>&lt;START T&gt;</td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>8</td>
<td></td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>&lt;T,A,8,16&gt;</td>
</tr>
<tr>
<td>READ(B,t)</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>t:=t*2</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>8</td>
<td>&lt;T,B,8,16&gt;</td>
</tr>
<tr>
<td>OUTPUT(A)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>OUTPUT(B)</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>&lt;COMMIT T&gt;</td>
</tr>
</tbody>
</table>

Can OUTPUT whenever we want: before/after COMMIT^{51}
Recovery with Undo/Redo Log

After system’s crash, run recovery manager

• Redo all committed transaction, top-down
• Undo all uncommitted transactions, bottom-up
Recovery with Undo/Redo Log

<START T1>
<T1,X1,v1>
<START T2>
<T2, X2, v2>
<START T3>
<T1,X3,v3>
<COMMIT T2>
<T3,X4,v4>
<T1,X5,v5>
...
...

Dan Suciu -- 444 Spring 2010
Granularity of the Log

• Physical logging: element = physical page
• Logical logging: element = data record

• What are the pros and cons?
Granularity of the Log

• Modern DBMS:

• Physical logging for the REDO part
  – Efficiency

• Logical logging for the UNDO part
  – For ROLLBACKs