Transactions and Recovery

Transactions

- A transaction is an action, or a series of actions, carried out by a single user or an application program, which reads or updates the contents of a database.

Atomicity and Consistency

- **Atomicity**
  - Transactions are atomic – they don’t have parts (conceptually)
  - can’t be executed partially; it should not be detectable that they interleave with another transaction

- **Consistency**
  - Transactions take the database from one consistent state into another
  - In the middle of a transaction the database might not be consistent

In This Lecture

- Transactions
- Recovery
  - System and Media Failures
- Concurrency
  - Concurrency problems
- For more information
  - Connolly and Begg chapter 20
  - Ullman and Widom 8.6

Isolation and Durability

- **Isolation**
  - The effects of a transaction are not visible to other transactions until it has completed
  - From outside the transaction has either happened or not
  - To me this actually sounds like a consequence of atomicity...

- **Durability**
  - Once a transaction has completed, its changes are made permanent
  - Even if the system crashes, the effects of a transaction must remain in place
Example of transaction

- Transfer £50 from account A to account B
  - Read(A)
  - A = A - 50
  - Write(A)
  - Read(B)
  - B = B + 50
  - Write(B)

Atomicity - shouldn’t take money from A without giving it to B
Consistency - money isn’t lost or gained
Isolation - other queries shouldn’t see A or B change until completion
Durability - the money does not go back to A

The Transaction Manager

- The transaction manager enforces the ACID properties
  - It schedules the operations of transactions
  - COMMIT and ROLLBACK are used to ensure atomicity
  - Locks or timestamps are used to ensure consistency and isolation for concurrent transactions (next lectures)
  - A log is kept to ensure durability in the event of system failure (this lecture)

COMMIT and ROLLBACK

- COMMIT signals the successful end of a transaction
  - Any changes made by the transaction should be saved
  - These changes are now visible to other transactions
- ROLLBACK signals the unsuccessful end of a transaction
  - Any changes made by the transaction should be undone
  - It is now as if the transaction never existed

Recovery

- Transactions should be durable, but we cannot prevent all sorts of failures:
  - System crashes
  - Power failures
  - Disk crashes
  - User mistakes
  - Sabotage
  - Natural disasters
  - Prevention is better than cure
  - Reliable OS
  - Security
  - UPS and surge protectors
  - RAID arrays
  - Can’t protect against everything though

The Transaction Log

- The transaction log records the details of all transactions
  - Any changes the transaction makes to the database
  - How to undo these changes
  - When transactions complete and how
  - The log is stored on disk, not in memory
    - If the system crashes it is preserved
  - Write ahead log rule
    - The entry in the log must be made before COMMIT processing can complete

System Failures

- A system failure means all running transactions are affected
  - Software crashes
  - Power failures
  - The physical media (disks) are not damaged
- At various times a DBMS takes a checkpoint
  - All committed transactions are written to disk
  - A record is made (on disk) of the transactions that are currently running
Transactions and Recovery

Types of Transactions

T1
T2
T3
T4
T5

Last Checkpoint  System Failure

Transactions and Recovery

System Recovery

• Any transaction that was running at the time of failure needs to be undone and restarted
• Any transactions that committed since the last checkpoint need to be redone
• Transactions of type T1 need no recovery
• Transactions of type T3 or T5 need to be undone and restarted
• Transactions of type T2 or T4 need to be redone

Transactions and Recovery

Transaction Recovery

UNDO and REDO: lists of transactions
UNDO = all transactions running at the last checkpoint
REDO = empty

For each entry in the log, starting at the last checkpoint
If a BEGIN TRANSACTION entry is found for T
Add T to UNDO
If a COMMIT entry is found for T
Move T from UNDO to REDO

Transactions and Recovery

Transaction Recovery

T1
T2
T3
T4
T5

Checkpoint Failure

UNDO: T2, T3
REDO:
Active transactions: T2, T3

Transactions and Recovery

Transaction Recovery

T1
T2
T3
T4
T5

Checkpoint Failure

UNDO: T2, T3, T4
REDO:
T4 Begins
Add T4 to UNDO

Transactions and Recovery

Transaction Recovery

T1
T2
T3
T4
T5

Checkpoint Failure

UNDO: T2, T3, T5
REDO:
T5 begins
Add T5 to UNDO

Transactions and Recovery
Transactions and Recovery

Forwards and Backwards
- Backwards recovery
  - We need to undo some transactions
  - Working backwards through the log we undo any operation by a transaction on the UNDO list
  - This returns the database to a consistent state
- Forwards recovery
  - Some transactions need to be redone
  - Working forwards through the log we redo any operation by a transaction on the REDO list
  - This brings the database up to date

Media Failures
- System failures are not too severe
  - Only information since the last checkpoint is affected
  - This can be recovered from the transaction log
- Media failures (disk crashes etc) are more serious
  - The data stored to disk is damaged
  - The transaction log itself may be damaged

Backups
- Backups are needed to recover from media failure
  - The transaction log and entire contents of the database is written to secondary storage (often tape)
  - Time consuming, and often requires down time
- Backups frequency
  - Frequent enough that little information is lost
  - Not so frequent as to cause problems
  - Every day (night) is common
- Backup storage

Recovery from Media Failure
- Restore the database from the last backup
- Use the transaction log to redo any changes made since the last backup
- If the transaction log is damaged you can't do step 2
  - Store the log on a separate physical device to the database
  - The risk of losing both is then reduced
Concurrency

- Large databases are used by many people
  - Many transactions to be run on the database
  - It is desirable to let them run at the same time as each other
  - Need to preserve isolation
- If we don’t allow for concurrency then transactions are run sequentially
  - Have a queue of transactions
  - Long transactions (e.g., backups) will make others wait for long periods

Concurrency Problems

- In order to run transactions concurrently we interleave their operations
- Each transaction gets a share of the computing time
- This leads to several sorts of problems
  - Lost updates
  - Uncommitted updates
  - Incorrect analysis
- All arise because isolation is broken

Lost Update

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X)</td>
<td>Read(X)</td>
</tr>
<tr>
<td>X = X - 5</td>
<td>X = X + 5</td>
</tr>
<tr>
<td>Write(X)</td>
<td>Write(X)</td>
</tr>
<tr>
<td>COMMIT</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

- T1 and T2 read X, both modify it, then both write it out
  - The net effect of T1 and T2 should be no change on X
  - Only T2’s change is seen, however, so the final value of X has increased by 5

Uncommitted Update

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X)</td>
<td>Read(X)</td>
</tr>
<tr>
<td>X = X - 5</td>
<td>X = X + 5</td>
</tr>
<tr>
<td>Write(X)</td>
<td>Write(X)</td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>COMMIT</td>
</tr>
</tbody>
</table>

- T2 sees the change to X made by T1, but T1 is rolled back
  - The change made by T1 is undone on rollback
  - It should be as if that change never happened

Inconsistent analysis

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read(X)</td>
<td>Read(X)</td>
</tr>
<tr>
<td>X = X - 5</td>
<td>X = X + 5</td>
</tr>
<tr>
<td>Write(X)</td>
<td>Write(X)</td>
</tr>
<tr>
<td>Read(Y)</td>
<td>Read(Y)</td>
</tr>
<tr>
<td>Y = Y + 5</td>
<td>Sum = X+Y</td>
</tr>
<tr>
<td>Write(Y)</td>
<td></td>
</tr>
</tbody>
</table>

- T1 doesn’t change the sum of X and Y, but T2 sees a change
  - T1 consists of two parts – take 5 from X and then add 5 to Y
  - T2 sees the effect of the first, but not the second

Next Lecture

- Concurrency
- Locks and resources
- Deadlock
- Serialisability
- Schedules of transactions
- Serial & serialisable schedules
- For more information
  - Connolly and Begg chapter 20
  - Ullman and Widom 8.6