Introduction to Transaction Management

CMPSCI 645
Mar 25, 2010

Slide content adapted from Ramakrishnan & Gehrke, Zack Ives
Concurrency Control

- Concurrent execution of user programs is essential for good DBMS performance
- We must also cope with partial operations
- The transaction is the foundation for:
  - Concurrent execution
  - Recovery from system failure, incomplete ops
What is a Transaction?

- A transaction is the DBMS’s abstract view of a user program: a sequence of reads and writes.
A simple transaction

- Imagine a simple banking application
  - Two database objects:
    - A: balance of account A
    - B: balance of account B
- Transaction T1:
  - “Transfer $100 from account B to account A”.

<table>
<thead>
<tr>
<th>T1: Transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin</td>
</tr>
<tr>
<td>A=A+100</td>
</tr>
<tr>
<td>B=B-100</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>
The ACID Properties

- Database systems ensure the ACID properties:
  - Atomicity
  - Consistency
  - Isolation
  - Durability
Atomicity

- A very important property guaranteed by the DBMS for all transactions is that they are atomic.
  - User can think of a Xact as executing all its actions in one step, or executing no actions at all.
  - DBMS logs all actions so that it can undo the actions of aborted transactions.
- If it succeeds, the effects of write operations persist (commit);
- If it fails, no effects of write operations persist (abort)
Consistency

- Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins.
  - DBMS will enforce some ICs, depending on the ICs declared in CREATE TABLE statements.
  - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
- In banking example, sum (A + B) should be unchanged by execution.
Isolation

- Many concurrent transactions are running at one time.
- Each transaction should be isolated from the effects of other transactions.
- Transactions should not be exposed to intermediate states created by other transactions.
- The net effect of concurrently running \{T1 and T2 and T3\} is equivalent to some serial order
  - No guarantee which serial order
Durability

- If transaction completes, its effects will persist in the database.
- In particular, if the system crashes before effects are written to disk, they will be redone.
- Recovery manager is responsible for this.
The ACID Properties

- Database systems ensure the ACID properties:
  - **Atomicity**: all operations of transaction reflected properly in database, or none are.
  - **Consistency**: each transaction in isolation keeps the database in a consistent state (this is the responsibility of the user).
  - **Isolation**: should be able to understand what’s going on by considering each separate transaction independently.
  - **Durability**: updates stay in the DBMS!!!
**Two transactions**

- "Transfer $100 from account B to account A"
- "Add 6% interest to accounts A and B"

<table>
<thead>
<tr>
<th>T1: Transfer</th>
<th>T2: Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin</td>
<td>Begin</td>
</tr>
<tr>
<td>A=A+100</td>
<td>A=1.06*A</td>
</tr>
<tr>
<td>B=B-100</td>
<td>B=1.06*B</td>
</tr>
<tr>
<td>End</td>
<td>End</td>
</tr>
</tbody>
</table>
Serial execution: T1, then T2

- Starting balances
  - A = 1000
  - B = 2000

- Execute T1
  - A = 1100
  - B = 1900

- Execute T2
  - A = 1166
  - B = 2014

<table>
<thead>
<tr>
<th>T1: Transfer</th>
<th>Begin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A=A+100</td>
</tr>
<tr>
<td></td>
<td>B=B-100</td>
</tr>
<tr>
<td></td>
<td>End</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>T2: Interest</th>
<th>Begin</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A=1.06*A</td>
</tr>
<tr>
<td></td>
<td>B=1.06*B</td>
</tr>
<tr>
<td></td>
<td>End</td>
</tr>
</tbody>
</table>
Serial execution: T2, then T1

- Starting balances
  - A = 1000
  - B = 2000
- Execute T2
  - A = 1060
  - B = 2120
- Execute T1
  - A = 1160
  - B = 2020

T2: Interest
Begin
A = 1.06*A
B = 1.06*B
End

T1: Transfer
Begin
A = A + 100
B = B - 100
End
Interleaved execution

- What other results are possible if operations of T1 and T2 are interleaved?
  - Starting balances
    - A = 1000
    - B = 2000

<table>
<thead>
<tr>
<th>T1: Transfer</th>
<th>T2: Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>A = A + 100</td>
<td>A = 1.06 * A</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>B = B - 100</td>
<td>B = 1.06 * B</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Interleaving operations

<table>
<thead>
<tr>
<th>T1: Transfer</th>
<th>T2: Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = A + 100</td>
<td>A = 1.06*A</td>
</tr>
<tr>
<td>B = B - 100</td>
<td>B = 1.06*B</td>
</tr>
</tbody>
</table>

Is this interleaving okay?
Interleaving operations

<table>
<thead>
<tr>
<th>T1: Transfer</th>
<th>T2: Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A = A + 100$</td>
<td>$A = 1.06 \times A$</td>
</tr>
<tr>
<td>$B = B - 100$</td>
<td>$B = 1.06 \times B$</td>
</tr>
</tbody>
</table>

How about this interleaving?
Goal: interleaved execution, with serial effects

- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect must be equivalent to these two transactions running serially in some order.
Scheduling Transactions

- A transaction is seen by DBMS as sequence of reads and writes
  - read of object O denoted R(O)
  - write of object O denoted W(O)
  - must end with Abort or Commit

- A schedule of a set of transactions is a list of all actions where order of two actions from any transaction must match order in that transaction.
A schedule

<table>
<thead>
<tr>
<th>T1: Transfer</th>
<th>T2: Interest</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = A + 100</td>
<td>A = 1.06 * A</td>
</tr>
<tr>
<td>B = B - 100</td>
<td>B = 1.06 * B</td>
</tr>
</tbody>
</table>

T1: Transfer
- Read(A)
- Write(A)

T2: Interest
- Read(A)
- Write(A)
- Read(B)
- Write(B)
- Read(B)
- Write(B)
Scheduling Transactions

- **Serial schedule**: Schedule that does not interleave the actions of different transactions.

- **Equivalent schedules**: For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.

- **Serializable schedule**: A schedule that is equivalent to some serial execution of the transactions.