Transactions (ACID)

Atomic
The all-or-nothing execution of transactions

Consistent
Transactions are expected to preserve the consistency and integrity of the database

Isolated
Transactions to be executed as if no other transaction is executing at the same time

Durable
Once a transactions has committed, its effects remain in the database
Transaction Management

Definition

Transaction = A list of writes and reads
For example: {Read, Read, Write}

Two Big Problems

1. Support multiple transaction at the same time
2. Make sure the data stored is reliable

Techniques

1. Concurrency Control
2. Database Recovery

Properties

Atomicity
Consistency
Isolation
Durability
Transaction Management

- Concurrency Control
  - Consistency
  - Isolation
- Database Recovery
  - Atomicity
  - Durability
Concurrency Control

- **Schedule**: Sequence of important actions taken by one or more transactions

From Chapter 18, The complete book
Scheduling

• A serial schedule is one that does not interleave the actions of different transactions

• A serializable schedule is a schedule that is equivalent to some serial schedule

• Two schedules are equivalent if, for any database state, the effect of executing them on database is identical
Example

From Chapter 18, The complete book

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ (A, t)</td>
<td></td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>t := t + 100</td>
<td></td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>WRITE (A, t)</td>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>READ (B, t)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t := t + 100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE (B, t)</td>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>READ (A, s)</td>
<td></td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>s := s * 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE (A, s)</td>
<td></td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>READ (B, s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>s := s * 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE (B, s)</td>
<td></td>
<td>250</td>
<td></td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>$T1$</th>
<th>$T2$</th>
<th>$A$</th>
<th>$B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A,s)</td>
<td></td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>$s:=s*2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(A,s)</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>READ(B,s)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$s:=s*2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(B,s)</td>
<td></td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>READ(A,t)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t:=t+100$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td></td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$t:=t+100$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td></td>
<td>150</td>
<td></td>
</tr>
</tbody>
</table>

From Chapter 18, The complete book
Example

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A,t)</td>
<td></td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>t:=t+100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(A,t)</td>
<td></td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>READ(A,s)</td>
<td>s:=s*2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(A,s)</td>
<td></td>
<td>250</td>
<td></td>
</tr>
<tr>
<td>READ(B,t)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t:=t+100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(B,t)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>READ(B,s)</td>
<td>s:=s*2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WRITE(B,s)</td>
<td></td>
<td>250</td>
<td></td>
</tr>
</tbody>
</table>

time
Concurrency Control

• The DBMS has freedom to interleave transactions

• Must pick an **interleaving** or **schedule** such that **isolation** and **consistency** are maintained → **Serializable schedule**

• If schedule is different than **any serial order**: **Not serializable**
Interleaving: Anomalies

• Various anomalies that break isolation and serializability occur because of (or with) certain conflicts between interleaved transactions

• Interleaving anomalies occur with or because of conflicts between transactions (conflicts can also occur without causing anomalies)
Anomalies

• Dirty read
  *(Occurring with / because of a \textit{WR conflict})*  
  \[ w_1(A) ; \ r_2(A) ; \ w_2(A) ; \ r_1(A) ; \]

  Changed A  
  Data written by a transaction that has not yet committed  
  \textbf{Dirty read} is a read of dirty data written by another transaction  
  \textbf{Risk}: the transaction that wrote it might \textbf{abort}

• Unrepeatable read
  *(Occurring with / because of a \textit{RW conflict})*  
  \[ r_1(A) ; \ r_2(A) ; \ w_2(A) ; \ r_1(A) ; \]

  Changed A

• Lost update
  *(Occurring because of a \textit{WW conflict})*  
  \[ w_1(A) ; \ w_2(A) ; \ w_2(B) ; \ w_1(B) ; \]

  Changed B  
  Changed B

12
Conflicts

• Two actions **conflict** if
  • They are part of **different transactions**
  • Involve the **same** variable
  • At least one of them is a **write**

• Three types of conflicts
  • Read-Write conflicts (**RW**)
  • Write-Read conflicts (**WR**)
  • Write-Write conflicts (**WW**)
Example

- Two actions **conflict** if
  - They are part of **different transactions**
  - Involve the **same** variable
  - At least one of them is a write

- What are all the conflicts here?
- Why?

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ(A)</td>
<td>WRITE(A)</td>
<td>READ(A)</td>
</tr>
<tr>
<td>WRITE(A)</td>
<td></td>
<td>WRITE(A)</td>
</tr>
<tr>
<td>READ(B)</td>
<td>WRITE(B)</td>
<td>READ(B)</td>
</tr>
<tr>
<td>WRITE(B)</td>
<td></td>
<td>WRITE(B)</td>
</tr>
</tbody>
</table>
Interleaving: Anomalies

- Conflict serializability is not necessary for serializability
Conflict Serializing

• Schedule $S$ is **conflict serializable** if $S$ is **conflict equivalent** to some serial schedule

• Two schedules are **conflict equivalent** if
  • They involve the **same actions** of the **same transactions**
  • *Every pair* of conflicting actions of two transactions are **ordered in the same way**
Example

- Conflict equivalent?

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ (A)</td>
<td>WRITE (A)</td>
<td>READ (A)</td>
<td>WRITE (A)</td>
</tr>
<tr>
<td>WRITE (A)</td>
<td>READ (B)</td>
<td>WRITE (A)</td>
<td>READ (B)</td>
</tr>
<tr>
<td>READ (B)</td>
<td>WRITE (B)</td>
<td>READ (B)</td>
<td>WRITE (B)</td>
</tr>
<tr>
<td>WRITE (B)</td>
<td></td>
<td>READ (A)</td>
<td>WRITE (A)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WRITE (A)</td>
<td>READ (B)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>READ (B)</td>
<td>WRITE (B)</td>
</tr>
</tbody>
</table>

Time
Example

- Conflict equivalent?

\[\begin{array}{c|c|c|c}
T1 & T2 & T1 & T2 \\
\hline
\text{READ (A)} & \text{READ (A)} & \text{READ (A)} & \text{READ (A)} \\
\text{WRITE (A)} & \text{WRITE (A)} & \text{WRITE (A)} & \text{WRITE (A)} \\
\text{READ (B)} & \text{READ (A)} & \text{READ (B)} & \text{READ (A)} \\
\text{WRITE (B)} & \text{WRITE (A)} & \text{WRITE (B)} & \text{WRITE (A)} \\
\end{array}\]
Example

- Conflict equivalent?

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ (A)</td>
<td>WRITE (A)</td>
</tr>
<tr>
<td>WRITE (A)</td>
<td>READ (A)</td>
</tr>
<tr>
<td>READ (B)</td>
<td>WRITE (B)</td>
</tr>
<tr>
<td>WRITE (B)</td>
<td>READ (A)</td>
</tr>
</tbody>
</table>

The diagram illustrates the sequence of operations for two transactions, T1 and T2, over time. The arrows indicate the sequence of actions, with read and write operations being executed in a top-down manner.
Conflict Graph

• **Conflict** Graph or **Precedence** Graph

• A graph with the **nodes as transactions**
  • There is an edge from $T_i \rightarrow T_j$ if any actions in $T_i$ precede and conflict with any actions in $T_j$

• **Theorem:** Schedule is **conflict serializable** if and only if its conflict graph is **acyclic**
Conflict Serializable?

- Find all conflicts
- Model the schedule as a conflict graph
- Check whether the graph has a cycle
  - Yes → not conflict serializable
  - No → conflict serializable
Example

- $r_1(A)$; $w_1(A)$; $r_2(A)$; $w_2(A)$; $r_2(B)$; $w_2(B)$; $r_1(B)$; $w_1(B)$
Example

- $r_2(A); r_1(B); w_2(A); r_3(A); w_1(B); w_3(A); r_2(B); w_2(B)$

- $r_2(A); r_1(B); w_2(A); r_2(B); r_3(A); w_1(B); w_3(A); w_2(B)$
Concurrency Control Algorithms

• Locking

• Timestamp Ordering
Database Recovery

• Failures
  • Erroneous Data Entry
    • Solution: Constraints and triggers

• Media Failures
  • RAID
  • Archive

• Catastrophic Failure
  • Archive
  • Redundant distributed copies

• System Failures
  • Logging
  • Checkpointing
Acknowledgements

I have used materials from the following resources in preparation of this course:

• **Database Systems: The Complete Book**
• Database Systems (Kifer, Bernstein, Lewis)
• Database System Concepts: [https://www.db-book.com](https://www.db-book.com)
• Course offerings
  • **CMPT 354 (Jiannan Wang - SFU):** [https://sfu-db.github.io/cmpt354/](https://sfu-db.github.io/cmpt354/)
  • W 4111 (Eugene Wu - Columbia): [https://w4111.github.io/](https://w4111.github.io/)
  • CS 186 (Joe Hellerstein - Berkeley): [https://sites.google.com/site/cs186fall17/](https://sites.google.com/site/cs186fall17/)
Transaction Management