CSE 344: Section 9
Transactions

May 24th, 2018
HW #8: Use Java + SQL Server to make application for booking flights
  • final homework of the quarter (yay!)
  • most time-consuming homework, so start early

Quiz #7: transactions & scheduling
Serializable

Conflict serializable is stricter than serializable

I.e. Any schedule that is conflict serializable must be serializable.
Serializability

Conflict serializable is stricter than serializable

I.e. Any schedule that is conflict serializable must be serializable.

Not all serializable schedules are conflict serializable:

<table>
<thead>
<tr>
<th>t1</th>
<th>t2</th>
</tr>
</thead>
<tbody>
<tr>
<td>W(A, 0)</td>
<td></td>
</tr>
<tr>
<td>R(A)</td>
<td>W(A, 0)</td>
</tr>
<tr>
<td></td>
<td>R(B)</td>
</tr>
</tbody>
</table>
Serializability

Checking for conflict serializability -> precedence graph and cycle checking
Serializability

S1: w1(Y); w2(Y); w1(X); w2(X); w3(X)

S2: w1(Y); w2(Y); w2(X); w1(X); w3(X)

Are these serializable?
Conflict serializable?
Serializability

S1: w1(Y); w2(Y); w1(X); w2(X); w3(X)

Conflict Serializable

S2: w1(Y); w2(Y); w2(X); w1(X); w3(X)

Serializable (but not conflict serializable)
2PL v.s. Strict 2PL

2PL:
- In every transaction, all lock requests must precede all unlock requests
- Ensure Conflict Serializability
- Might not be able to recover (Dirty Read: Read on some write that gets rolled back)

Strict 2PL:
- Every lock each transaction is held until commit or abort
- Ensure Conflict Serializability
- Recoverable as each transaction does not affect others until commit/abort
A New Problem: Non-recoverable Schedule

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_1(A); L_1(B); READ(A)</td>
<td>L_2(A); READ(A)</td>
</tr>
<tr>
<td>A := A + 100</td>
<td>A := A * 2</td>
</tr>
<tr>
<td>WRITE(A); U_1(A)</td>
<td>WRITE(A);</td>
</tr>
<tr>
<td>READ(B)</td>
<td>L_2(B); BLOCKED...</td>
</tr>
<tr>
<td>B := B + 100</td>
<td>...GRANTED; READ(B)</td>
</tr>
<tr>
<td>WRITE(B); U_1(B);</td>
<td>B := B * 2</td>
</tr>
<tr>
<td></td>
<td>WRITE(B); U_2(A); U_2(B);</td>
</tr>
<tr>
<td></td>
<td>Commit</td>
</tr>
<tr>
<td>Rollback</td>
<td></td>
</tr>
</tbody>
</table>
Isolation Level: Read Uncommitted

Write Locks? Strict 2PL

Read Locks? No (Immediate Read)

Problem: Dirty-Read

Reading uncommitted data that can be rolled back
Isolation Level: Read Uncommitted

Example Transaction:

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W(A)</td>
<td>R(A)</td>
</tr>
<tr>
<td></td>
<td>W(B)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Commit</td>
</tr>
<tr>
<td></td>
<td>R(B)</td>
<td>Commit</td>
</tr>
</tbody>
</table>

T2 is reading value of A updated by T1’s write on A, but T1 has not committed yet.

The value of A read by T2 might not even be in the result.

Then T2’s action can be influenced by such uncommitted data.
Isolation Level: Read Committed

Write Locks?  Strict 2PL

Read Locks? Obtain before read, release after (No more dirty read)

Problem: Unrepeatable Read

The values of 2 reads on the same tuple can be different in the same transaction
Isolation Level: Read Committed

Example Transaction:

\[
S = \begin{bmatrix}
T1 & T2 \\
R(A) & R(A) \\
R(A) & W(A) \\
W(A) & Com. \\
R(A) & Com.
\end{bmatrix}
\]

T1’s first R(A) and T1’s second R(A) might have different results.

Updated by T2’s W(A).
Isolation Level: Repeatable Read

**Write Locks?** Strict 2PL

**Read Locks?** Strict 2PL (No more unrepeatable read)

Same as Serializable if no insert or delete

**Problem:** Phantom Read

In the same transaction, some tuples appear sometimes and disappear other times
Isolation Level: Repeatable Read

Suppose there are two blue products, A1, A2:

**Phantom Problem**

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT *</td>
<td>INSERT INTO Product(name, color) VALUES ('A3','blue')</td>
</tr>
<tr>
<td>FROM Product</td>
<td></td>
</tr>
<tr>
<td>WHERE color='blue'</td>
<td></td>
</tr>
</tbody>
</table>

SELECT *
FROM Product
WHERE color='blue'
Isolation Level: Serializable

Not the same thing as Serializable schedule!!

Write Locks: Strict 2PL

Read Locks: Strict 2PL

Predicate Lock/Table Lock (No Phantom)

Difference between Repeatable Read and Serializable is that serializable schedule blocks inserts & deletes from another transaction
Isolation Level: Serializable

Predicate Lock Example:

In Transaction T, we have a statement:

```
SELECT * FROM People WHERE age > 18;
```

In this case, the transaction will grab a predicate lock that prevent inserting and deleting tuples that can affect the predicate/statement.

In this case, the lock prevents inserting and deleting tuples with age > 18.
## Isolation Level: Summary

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Dirty Reads</th>
<th>Nonrepeatable Reads</th>
<th>Phantom Inserts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Uncommitted</td>
<td>Allowed</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>Read Committed</td>
<td>Not Allowed</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>Repeatable Read</td>
<td>Not Allowed</td>
<td>Not Allowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>Serializable</td>
<td>Not Allowed</td>
<td>Not Allowed</td>
<td>Not Allowed</td>
</tr>
</tbody>
</table>