Transactions 2

CS348 Spring 2023
Instructor: Sujaya Maiyya
Sections: 002 & 004 only
Outline For Today

1. Motivation For Transactions
2. ACID Properties
3. Different Levels of Isolation Beyond Serializability

Serializability:
- Execution Histories
- Conflict Equivalence
- Checking For Conflict Equivalence

Concurrency control

Last lecture:
User’s Perspective

Today’s lecture:
System’s Perspective
Concurrency is achieved by interleaving operations across txns.

Q: Does an interleaving correspond to a serializable execution?

Execution history model and conflict equivalences is a formal method to answer this question.
Representing Single Transactions

- Database is a set of *data items* (often will denote as x, y, z...)
- Txn $T_i$ is a *total order* of read/write & commit/abort operations on items
  - $r_i(x)$ indicates $T_i$ reads item x
  - $w_i(x)$ indicates $T_i$ writes item x
  - $c$ indicates commit (a indicates aborts)

  - Suppose: $T_i$ does the following in this *chronological order*:
    - Read(x), Read(y), $x \leftarrow x + y$, Write(x), commit
    - $T_i = \{r_i(x) < r_i(y) < w_i(x) < c_i\}$ or simply as:
      - $T_i = \{r_i(x), r_i(y), w_i(x), c_i\}$ or $r_i(x), r_i(y), w_i(x), c_i$

- DAG representation

```
  r(x) -----> w(x) -----> c
     |          |          
  r(y) ----> w(x)
```
Execution histories (or schedules)

- An execution history over a set of transactions $T_1 \ldots T_n$ is an interleaving of the operations of $T_1 \ldots T_n$ in which the operation ordering imposed by each transaction is preserved.

- Two important assumptions:
  - Transactions interact with each other only via reads and writes of objects
  - A database is a fixed set of independent objects

- Example: $T_1 = \{w_1[x], w_1[y], c_1\}$, $T_2 = \{r_2[x], r_2[y], c_2\}$
  - $H_a = w_1[x]r_2[x]w_1[y]r_2[y]c_1c_2$
  - $H_b = w_1[x]w_1[y]c_1r_2[x]r_2[y]c_2$
  - $H_c = w_1[x]r_2[x]r_2[y]w_1[y]c_1c_2$ [next slide expands this example]
  - $H_d = r_2[x]r_2[y]c_2 w_1[x]w_1[y]c_1$
Examples for valid execution history

- $T_1 = \{ w_1[x], w_1[y], c_1 \}$, $T_2 = \{ r_2[x], r_2[y], c_2 \}$

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1(x)$</td>
<td>$w_1(x)$</td>
</tr>
<tr>
<td>$r_2(x)$</td>
<td>$w_1(y)$</td>
</tr>
<tr>
<td>$w_1(y)$</td>
<td>$c_1$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$c_2$</td>
<td></td>
</tr>
</tbody>
</table>

$H_a$  $H_b$  $H_c$  $H_d$
Examples for invalid execution history

\[ T_1 = \{ w_1[x], w_1[y], c_1 \}, T_2 = \{ r_2[x], r_2[y], c_2 \} \]

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Incorrect orders

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Incorrect orders
Serial execution histories

- \( T_1 = \{ w_1[x], w_1[y], c_1 \}, T_2 = \{ r_2[x], r_2[y], c_2 \} \)

- Serial histories: no interleaving operations from different txns

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c}
T_1 & T_2 & T_1 & T_2 & T_1 & T_2 & T_1 & T_2 & T_1 & T_2 \\
\hline
\phantom{w_1}(x) & w_1(x) & w_1(x) & w_1(x) & w_1(x) & w_1(x) & w_1(x) & w_1(x) & w_1(x) & w_1(x) \\
\phantom{w_1}(y) & r_2(x) & w_1(y) & r_2(x) & w_1(y) & r_2(x) & w_1(y) & r_2(x) & w_1(y) & r_2(x) \\
c_1 & c_1 & r_2(y) & r_2(y) & r_2(y) & r_2(y) & r_2(y) & r_2(y) & r_2(y) & r_2(y) \\
c_2 & c_2 & c_2 & c_2 & c_2 & c_2 & c_2 & c_2 & c_2 & c_2 \\
\end{array}
\]
Equivalent histories

• $H_a$ is “equivalent” to $H_b$ (a serial execution)

$T_1$  |  $T_2$
---|---
Write 4 | Write 4
$w_1(x)$ | $w_1(x)$
Write 5 | Write 5
$w_1(y)$ | $w_1(y)$
Read 4 | Read 4
$w_1(y)$ | $r_2(y)$
Read 5 | Read 5

$T_1$  |  $T_2$
---|---
$w_1(x)$ | $w_1(x)$
$w_1(y)$ | $w_1(y)$
c1 | c1
$w_1(y)$ | $r_2(y)$
c2 | c2

$T_2$ sees all the updates made by $T_1$
- $T_2$ reads $x$ written by $T_1$
- $T_2$ reads $y$ written by $T_1$

$x=3$, $y=1$ (before $T_1$ and $T_2$)
Equivalent histories

- \( H_c \) is not “equivalent” to \( H_b \) (a serial execution)

\[
\begin{align*}
T_1 & \quad T_2 \\
\text{Write 4} & \quad \text{Write 5} \\
\text{Write 5} & \quad \text{Write 5} \\
\text{Read 4} & \quad \text{Read 5} \\
\text{Read 5} & \quad \text{Read 1} \\
x=3, \ y=1 & \quad (\text{before } T_1 \text{ and } T_2)
\end{align*}
\]
Outline For Today

Serializability:

1. Execution Histories
2. Conflict Equivalence
3. Checking For Serializability

Concurrency control:

1. 2 phase locking
Check equivalence

• Two operations conflict if:
  1. they belong to different transactions,  
  2. they operate on the same object, and  
  3. at least one of the operations is a write

2 types of conflicts: (1) Read-Write (or write-read) and (2) Write-Write

• Two histories are (conflict) equivalent if
  1. they are over the same set of transactions, and 
  2. the ordering of each pair of conflicting operations is the same in each history
Example

• Consider
  • $H_a = w_1[x]r_2[x]w_1[y]r_2[y]c_1c_2$
  • $H_b = w_1[x]w_1[y]r_2[x]r_2[y]c_1c_2$

Step 1: check if they are over the same set of transactions
  • $T_1 = \{w_1[x], w_1[y]\}$, $T_2 = \{r_2[x], r_2[y]\}$

Step 2: check if all the conflicting pairs have the same order

<table>
<thead>
<tr>
<th>Conflicting pairs</th>
<th>$H_a$</th>
<th>$H_b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1[x], r_2[x]$</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>$w_1[y], r_2[y]$</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>
Motivation & Intuition For Conflict Equivalence

- If two histories $H_a$ and $H_b$ are conflict equivalent then, we can make $H_a$ exactly the same as $H_b$ by iteratively swapping two consecutive non-conflicting operations in $H_a$ and/or $H_b$.
  - $H_a = w_1[x]r_2[x]w_1[y]r_2[y]c_1c_2 \Rightarrow H'_a = w_1[x]w_1[y]r_2[x]r_2[y]c_1c_2$
  - $H_b = w_1[x]w_1[y]r_2[x]r_2[y]c_1c_2$

- Proof Sketch: Move all ops on item $x$ to the beginning by swapping with non-conflicting ops in both $H_a$ and $H_b$
- End with the order imposed by the conflicts on $x$
- If $H_a$ & $H_b$ are conflict eq. this prefix ops on $x$ will be the same order
- Then repeat for $y$, $z$, etc. and we will arrive at the same histories
- Therefore: Every read by each txn has the same value in $H_a$ & $H_b$
- Therefore: $H_a$ & $H_b$ lead to the same output database state.
More complicated example

Consider

• $H_A$: $r_1[x]r_3[x]w_4[y]r_2[u]w_4[z]r_1[y]r_3[u]r_2[z]w_2[z]r_3[z]r_1[z]w_3[y]$
• $H_B$: $r_1[x]w_4[y]r_3[x]r_2[u]r_1[y]r_3[u]r_2[z]w_2[z]w_4[z]r_1[z]r_3[z]w_3[y]$

Step 1: check if they are over the same set of transactions

Step 2: check if all the conflicting pairs have the same order
More complicated example

Consider

• $H_A$: $r_1[x]r_3[x]w_4[y]r_2[u]w_4[z]r_1[y]r_3[u]r_2[z]w_2[z]r_3[z]r_1[z]w_3[y]

• $H_B$: $r_1[x]w_4[y]r_3[x]r_2[u]r_1[y]r_3[u]r_2[z]w_2[z]w_4[z]r_1[z]r_3[z]w_3[y]

Step 1: check if they are over the same set of transactions

\{r_1[x] r_1[y] r_1[z]\}, \{r_2[u] r_2[z]w_2[z]\}, \{r_3[x] r_3[u] r_3[z]w_3[y]\}, \\
\{w_4[y] w_4[z]\}

Step 2: check if all the conflicting pairs have the same order
Identify all the conflicting pairs

• $H_A$: $r_1[x]r_3[x]w_4[y]r_2[u]w_4[z]r_1[y]r_3[u]r_2[z]w_2[z]r_3[z]r_1[z]w_3[y]$

• Conflicting pairs:
  • Related to $x$ or $u$: no conflicting pairs, as all are reads
  • Related to $y$: $w_4[y]$, $r_1[y]$, $w_3[y]$
    • $w_4[y] < r_1[y]$
    • $w_4[y] < w_3[y]$
    • $r_1[y] < w_3[y]$
  • Related to $z$: $w_4[z]$, $r_2[z]$, $w_2[z]$, $r_3[z]$, $r_1[z]$
    • $w_4[z] < r_2[z]$
    • $w_4[z] < w_2[z]$
    • $w_4[z] < r_3[z]$
    • $w_4[z] < r_1[z]$
    • $r_2[z]$, $w_2[z]$ are not, as they are from the same transactions
    • $w_2[z] < r_3[z]$
    • $w_2[z] < r_1[z]$
More complicated example

Consider

- $H_A$: $r_1[x]r_3[x]w_4[y]r_2[u]w_4[z]r_1[y]r_3[u]r_2[z]w_2[z]r_3[z]r_1[z]w_3[y]$
- $H_B$: $r_1[x]w_4[y]r_3[x]r_2[u]r_1[y]r_3[u]r_2[z]w_2[z]w_4[z]r_1[z]r_3[z]w_3[y]$

Step 1: check if they are over the same set of transactions

\{r_1[x], r_1[y], r_1[z]\}, \{r_2[u], r_2[z], w_2[z]\}, \{r_3[x], r_3[u], r_3[z], w_3[y]\}, \{w_4[y], w_4[z]\}

Step 2: check if all the conflicting pairs have the same order

<table>
<thead>
<tr>
<th>Conflicting pairs</th>
<th>$H_A$</th>
<th>$H_B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_4[y], r_1[y]$</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>$w_4[y], w_3[y]$</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>...</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
<tr>
<td>$w_4[z], w_2[z]$</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>...</td>
<td>&lt;</td>
<td>&lt;</td>
</tr>
</tbody>
</table>
Outline For Today

Serializability:

1. Execution Histories
2. Conflict Equivalence
3. Checking For Serializability

Concurrency control:

1. 2 phase locking
A history $H$ is said to be (conflict) **serializable** if there exists some serial history $H'$ that is (conflict) equivalent to $H$.

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1(x)$</td>
<td>$r_2(x)$</td>
<td>$w_1(x)$</td>
<td>$w_1(y)$</td>
<td>$r_2(x)$</td>
<td>$r_2(y)$</td>
</tr>
<tr>
<td>$w_1(y)$</td>
<td>$r_2(y)$</td>
<td>$c_1$</td>
<td>$r_2(y)$</td>
<td>$c_1$</td>
<td>$c_2$</td>
</tr>
<tr>
<td>$c_1$</td>
<td>$c_2$</td>
<td>$H_a$</td>
<td>$H_b$</td>
<td>$H_c$</td>
<td></td>
</tr>
</tbody>
</table>
Serializable

• Does $H_c$ have an equivalent serial execution?
  • $H_c = w_1[x]r_2[x]r_2[y]w_1[y]c_1c_2$

• Only 2 serial execution to check:
  • $H_b$: $T_1$ followed by $T_2$: $w_1[x]w_1[y]c_1r_2[x]r_2[y]c_2$
    • $r_2[y]$ reads different value as in $H_c$
  • $H_d$: $T_2$ followed by $T_1$: $r_2[x]r_2[y]c_2w_1[x]w_1[y]c_1$
    • $r_2[x]$ reads different value as in $H_c$

<table>
<thead>
<tr>
<th>Conflicting pairs</th>
<th>$H_b$</th>
<th>$H_c$</th>
<th>$H_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_1[x], r_2[x]$</td>
<td>&lt;</td>
<td>&lt;</td>
<td>&gt;</td>
</tr>
<tr>
<td>$w_1[y], r_2[y]$</td>
<td>&lt;</td>
<td>&gt;</td>
<td>&gt;</td>
</tr>
</tbody>
</table>

• Do we need to check all the serial executions?
How to test for serializability?

• Serialization graph $SG_H(V, E)$ for history $H$:
  • $V = \{ T | T $ is a committed transaction in $H \}$
  • $E = \{ T_i \rightarrow T_j $ if $o_i \in T_i $ and $o_j \in T_j $ conflict and $o_i < o_j \}$

• A history is serializable iff its serialization graph is acyclic.
Example

- Example: $H_a = w_1[x]r_2[x]w_1[y]r_2[y]c_1c_2$

$w_1[x] \text{ and } r_2[x] \text{ conflict, and } w_1[x] < r_2[x]$

$w_1[y] \text{ and } r_2[y] \text{ conflict, and } w_1[y] < r_2[y]$

Serialization graph: no cycles $\rightarrow$ serializable
Example

- Example: $H_c = w_1[x]r_2[x]r_2[y]w_1[y]c_1c_2$

$w_1[x]$ and $r_2[x]$ conflict, and $w_1[x] < r_2[x]$;

$w_1[y]$ and $r_2[y]$ conflict, and $r_2[y] < w_1[y]$
More complicated example

- \( r_1[x]r_3[x]w_4[y]r_2[u]w_4[z]r_1[y]r_3[u]r_2[z]w_2[z]r_3[z]r_1[z]w_3[y] \)

- Conflicting pairs:
  - Related to \( x \) or \( u \): no conflicting pairs, as all are reads
  - Related to \( y \): \( w_4[y], r_1[y], w_3[y] \)
    - \( w_4[y] < r_1[y] \quad T_4 \rightarrow T_1 \)
    - \( w_4[y] < w_3[y] \quad T_4 \rightarrow T_3 \)
    - \( r_1[y] < w_3[y] \quad T_1 \rightarrow T_3 \)
  - Related to \( z \): \( w_4[z], r_2[z], w_2[z], r_3[z], r_1[z] \)
    - \( w_4[z] < r_2[z] \quad T_4 \rightarrow T_2 \)
    - \( w_4[z] < w_2[z] \quad T_4 \rightarrow T_2 \)
    - \( w_4[z] < r_3[z] \quad T_4 \rightarrow T_3 \)
    - \( w_4[z] < r_1[z] \quad T_4 \rightarrow T_1 \)
    - \( r_2[z], w_2[z] \) are not, as they are from the same transactions
    - \( w_2[z] < r_3[z] \quad T_2 \rightarrow T_3 \)
    - \( w_2[z] < r_1[z] \quad T_2 \rightarrow T_1 \)
More complicated example

- $r_1[x]r_3[x]w_4[y]r_2[u]w_4[z]r_1[y]r_3[u]r_2[z]w_2[z]r_3[z]r_1[z]w_3[y]$

- No cycles in this serialization graph
  - Topological sort: T4 -> T2 -> T1 -> T3

- The history above is (conflict) equivalent to
  - $w_4[y]w_4[z]r_2[u]r_2[z]w_2[z]r_1[x]r_1[y]r_1[z]r_3[z]r_3[x]r_3[u]r_3[z]w_3[y]$
  - Note: we ignore the commits at the end for simplicity
Outline For Today

Serializability:
1. Execution Histories
2. Conflict Equivalence
3. Checking For Serializability

Concurrency control:
1. 2 phase locking
Concurrency control

- Goal: ensure the “I” (isolation) in ACID
Good versus bad execution histories

<table>
<thead>
<tr>
<th>Serial</th>
<th>Good!</th>
<th>Good! Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>$T_2$</td>
<td>$T_1$</td>
</tr>
<tr>
<td>$T_2$</td>
<td></td>
<td>$T_2$</td>
</tr>
<tr>
<td>r1(x)</td>
<td>r1(x)</td>
<td>r1(x)</td>
</tr>
<tr>
<td>w1(x)</td>
<td>w1(x)</td>
<td>w1(x)</td>
</tr>
<tr>
<td>r1(y)</td>
<td>r1(y)</td>
<td>w2(x)</td>
</tr>
<tr>
<td>w1(y)</td>
<td>w1(y)</td>
<td>w1(y)</td>
</tr>
<tr>
<td>r2(x)</td>
<td>r2(x)</td>
<td>r2(x)</td>
</tr>
<tr>
<td>w2(x)</td>
<td>w2(x)</td>
<td>w2(x)</td>
</tr>
<tr>
<td>r2(z)</td>
<td>r2(z)</td>
<td>r2(z)</td>
</tr>
<tr>
<td>w2(z)</td>
<td>w2(z)</td>
<td>w2(z)</td>
</tr>
</tbody>
</table>

$H_a$ | $H_b$ | $H_c$ |

Read 400
Write 400 – 100

Read 400
Write 400 – 50

Write

Hint: construct serialization graph
Good versus bad execution histories

Not serializable

Bad!

How to avoid this?

Note: These are ‘valid’ histories but are ‘bad’: cannot be serialized
Concurrency control

Possible classification

- **Pessimistic** – assume that conflicts will happen and take preventive action
  - Two-phase locking (2PL)

- **Optimistic** – assume that conflicts are rare and run transactions and fix if there is a problem
  - Timestamp ordering

- We will only review 2PL
Locking

• Rules
  • If a transaction wants to read an object, it must first request a shared lock (S mode) on that object
  • If a transaction wants to modify an object, it must first request an exclusive lock (X mode) on that object
  • Allow one exclusive lock, or multiple shared locks

Mode of lock(s) currently held by other transactions

<table>
<thead>
<tr>
<th>Mode of lock(s) currently held</th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>X</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Compatibility matrix

Mode of the lock requested

<table>
<thead>
<tr>
<th>Mode of the lock requested</th>
<th>S</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>X</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Grant the lock?
Basic locking is not enough

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td>lock-X(x)</td>
<td>lock-X(x)</td>
</tr>
<tr>
<td>r1(x)</td>
<td>r2(x)</td>
</tr>
<tr>
<td>w1(x)</td>
<td>w2(x)</td>
</tr>
<tr>
<td>unlock(x)</td>
<td>unlock(x)</td>
</tr>
</tbody>
</table>

Possible schedule under locking

But still not conflict-serializable!
Basic locking is not enough

Add 1 to both x and y (preserve x=y)
Read 100
Write 100+1

Possible schedule under locking

But still not conflict-serializable!

lock-X(x)
lock-X(y)
lock-X(x)
lock-X(y)

lock-X(x)
lock-X(y)
lock-X(x)
lock-X(y)

unlock(x)
unlock(x)
unlock(x)
unlock(x)

Read 100
Read 101
Read 100
Read 100

Write 100+1
Write 101*2
Write 100
Write 100*2

Multiplying both x and y by 2 (preserves x=y)

Read 200
Read 200

x ≠ y!
Two-phase locking (2PL)

- All lock requests precede all unlock requests
  - Phase 1: obtain locks, phase 2: release locks

$T_1$ $T_2$

lock-X(x)
lock-X(y)
unlock(x)
unlock(y)

Cannot obtain the lock on y until $T_1$ unlocks

$T_1$ $T_2$

r1(x)
w1(x)
r1(y)
w1(y)
r2(x)
w2(x)
r2(y)
w2(y)

2PL guarantees a conflict-serializable schedule
Remaining problems of 2PL

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1(x)</td>
<td>r2(x)</td>
</tr>
<tr>
<td>r1(y)</td>
<td>r2(y)</td>
</tr>
<tr>
<td>w1(x)</td>
<td>w2(x)</td>
</tr>
<tr>
<td>w1(y)</td>
<td>w2(y)</td>
</tr>
<tr>
<td>Abort!</td>
<td></td>
</tr>
</tbody>
</table>

- $T_2$ has read uncommitted data written by $T_1$
- If $T_1$ aborts, then $T_2$ must abort as well
- Cascading aborts possible if other transactions have read data written by $T_2$

- Even worse, what if $T_2$ commits before $T_1$?
  - Schedule is not recoverable if the system crashes right after $T_2$ commits
Deadlocks

• A transaction is deadlocked if it is blocked and will remain blocked until there is an intervention.

• Locking-based concurrency control algorithms may cause deadlocks requiring abort of one of the transactions.

• Consider the partial history
  • Neither $T_1$ nor $T_2$ can make progress

  Cannot obtain the lock on $y$ until $T_2$ unlocks

  Cannot obtain the lock on $y$ until $T_1$ unlocks
Strict 2PL

• Only release X-locks at commit/abort time
  • A writer will block all other readers until the writer commits or aborts

• Used in many commercial DBMS
  • Avoids cascading aborts
  • But deadlocks are still possible!

• Conservative 2PL: acquire all locks at the beginning of a txn
  • Avoids deadlocks but often not practical
Summary

Serializability:

1. Execution Histories
2. Conflict Equivalence
3. Checking For Serializability

Concurrency control:

1. 2 phase locking