Transactions & Update Correctness
Correctness

• Data Correctness (Constraints)

• Query Correctness (Plan Rewrites)

• Update Correctness (Transactions)
What could go wrong?

- **Parallelism**: What happens if two updates modify the same data?
- Maximize use of IO / Minimize Latencies.

- **Persistence**: What happens if something breaks during an update?
- When is my data safe?
What does it mean for a database operation to be correct?
What is an Update?

• INSERT INTO ...?
• UPDATE ... SET ... WHERE ...?
• Non-SQL?

Can we abstract?
Abstract Update Operations

[Transaction]

Read → Read → Read
Write → Write → Write
Abort
Commit

Time
What does it mean for a database operation to be correct?
Transaction Correctness

- Reliability in database transactions guaranteed by ACID

- A - Atomicity (“Do or Do Not, there is nothing like try”) - usually ensured by logs

- C - Consistency (“Within the framework of law”) - usually ensured by integrity constraints, validations, etc.

- I - Isolation (“Execute in parallel or serially, the result should be same”) - usually ensured by locks

- D - Durability (“once committed, remain committed”) - usually ensured at hardware level
Atomicity

- A transaction completes by committing, or terminates by aborting.

- **Logging** is used to undo aborted transactions.

- **Atomicity**: A transaction is (or appears as if it were) applied in one ‘step’, independent of other transactions.

- All ops in a transaction commit or abort together.
Isolation

T1: BEGIN A=A+100, B=B−100 END  
T2: BEGIN A=1.06*A, B=1.06*B END

• Intuitively, T1 transfers $100 from A to B and T2 credits both accounts with interest.

• What are possible interleaving errors?
Example: Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A=A+100</td>
<td>A=1.06*A</td>
</tr>
<tr>
<td></td>
<td>B=B−100</td>
<td>B=1.06*B</td>
</tr>
</tbody>
</table>

OK!
Example: Schedule

<table>
<thead>
<tr>
<th>Time</th>
<th>T1</th>
<th>T2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A = A + 100</td>
<td>A = 1.06 * A</td>
</tr>
<tr>
<td></td>
<td>B = B − 100</td>
<td>B = 1.06 * B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not OK!</td>
</tr>
</tbody>
</table>
Example: The DBMS’s View

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

Not OK!
What went wrong?
What could go wrong?

Reading uncommitted data
(write-read/WR conflicts; aka “Dirty Reads”)

T1: \(R(A), W(A), \) \(R(B), W(B), \) ABRT
T2: \(R(A), W(A), \) CMT,

Unrepeatable Reads
(read-write/RW conflicts)

T1: \(R(A), \) \(R(A), W(A), \) CMT
T2: \(R(A), W(A), \) CMT,
What could go wrong?

Overwriting Uncommitted Data
(write-write/WW conflicts)

T1: W(A), W(B), CMT
T2: W(A), W(B), CMT,
Schedule
An ordering of read and write operations.

Serial Schedule
No interleaving between transactions at all

Serializable Schedule
Guaranteed to produce equivalent output to a serial schedule
Conflict Equivalence

Possible Solution: Look at read/write, etc… conflicts!

Allow operations to be reordered as long as conflicts are ordered the same way.

Conflict Equivalence: Can reorder one schedule into another without reordering conflicts.

Conflict Serializability: Conflict Equivalent to a serial schedule.
Conflict Serializability

• **Step 1:** Serial Schedules are *Always Correct*

• **Step 2:** Schedules with the same operations and the same conflict ordering are *conflict-equivalent*.

• **Step 3:** Schedules *conflict-equivalent* to an always correct schedule are also correct.

• … or *conflict serializable*
Example

Time

T1

W(B)

R(B)

W(A)

R(A)

Conflict

T2

W(B)

R(B)

R(A)

W(A)

VS.

T1

W(B)

R(B)

R(A)

W(A)
Example

Time

\[
\begin{align*}
R(B) & \quad 1 \quad W(B) \\
W(A) & \quad 2 \quad R(A)
\end{align*}
\]

\[1: T2 \rightarrow T1 \quad 2: T1 \rightarrow T2\]

\[\neq\]

\[
\begin{align*}
R(B) & \quad 1 \quad W(B) \\
W(A) & \quad 2 \quad R(A)
\end{align*}
\]

\[1: T2 \rightarrow T1 \quad 2: T2 \rightarrow T1\]
Equivalence

- Look at the actual effects
  - Can’t determine effects without running
- Look at the conflicts
  - Too strict
- Look at the possible effects
Information Flow

Old State

T1

New State

R(...)
Information Flow
Information Flow

Multiple Transactions

R(…)

R(…)

R(…)

R(…)

R(…)

R(…)

R(…)

R(…)
View Serializability

**Possible Solution**: Look at data flow!

**View Equivalence**: All reads read from the same writer
Final write in a batch comes from the same writer

**View Serializability**: Conflict Equivalent to a serial schedule.
View Equivalence

- For all Reads \( R \)
  - If \( R \) reads old state in \( S_1 \), \( R \) reads old state in \( S_2 \)
  - If \( R \) reads \( T_i \)'s write in \( S_1 \), \( R \) reads the same write in \( S_2 \)
- For all values \( V \) being written.
  - If \( W \) is the last write to \( V \) in \( S_1 \), \( W \) is the last write to \( V \) in \( S_2 \)
- If these conditions are satisfied, \( S_1 \) and \( S_2 \) are view-equivalent
View Serializability

• **Step 1:** Serial Schedules are *Always Correct*

• **Step 2:** Schedules with the same information flow are *view-equivalent.*

• **Step 3:** Schedules *view-equivalent* to an always correct schedule are also correct.

  • … or *view serializable*
Example

Time

R(A)

W(A)

W(A)

W(A)
Example

Write order irrelevant
(T3 overwrites either way)
Enforcing Serializability

- Conflict Serializability:
  - Does locking enforce conflict serializability?

- View Serializability
  - Is view serializability stronger, weaker, or incomparable to conflict serializability?

- What do we need to enforce either fully?
How to detect conflict serializable schedule?

<table>
<thead>
<tr>
<th></th>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W(a)</td>
<td>R(b)</td>
<td>W(d)</td>
</tr>
<tr>
<td></td>
<td>W(b)</td>
<td>R(d)</td>
<td>W(d)</td>
</tr>
</tbody>
</table>

Precedence Graph:

- Cycle! Not Conflict serializable

It is not conflict serializable because the precedence graph has a cycle.

Every non-serializable schedule cannot be 2PL or strict 2PL.

It is not serializable because of a cycle in the precedence graph.

It can not be strict 2PL for the same reasons with the first schedule.
Not conflict serializable but view serializable

Satisfies 3 conditions of view serializability

Every view serializable schedule which is not conflict serializable has blind writes.
How can conflicts be avoided?

- Optimistic Concurrency Control
- Conservative Concurrency Control
Conservative Concurrency Control

• How can bad schedules be detected?

• What problems does each approach introduce?

• How do we resolve these problems?
Two-Phase Locking

- Phase 1: Acquire (do not release) locks.
- Phase 2: Release (do not acquire) locks.

Why?

Can we do even better?
Example

Acyclic - Conflict Serializable
2PL exists
Example

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td>L(a)</td>
<td>L(b)</td>
<td>L(d)</td>
</tr>
<tr>
<td>W(a)</td>
<td>R(b)</td>
<td>R(d)</td>
</tr>
<tr>
<td></td>
<td>W(d)</td>
<td>W(d)</td>
</tr>
<tr>
<td></td>
<td>R-L(b)</td>
<td>R-L(d)</td>
</tr>
<tr>
<td></td>
<td>L(d)</td>
<td>R-L(b)</td>
</tr>
<tr>
<td></td>
<td>L(b)</td>
<td>R(d)</td>
</tr>
<tr>
<td></td>
<td>R-L(a)</td>
<td>R-L(d)</td>
</tr>
<tr>
<td></td>
<td>W(b)</td>
<td>R-L(b)</td>
</tr>
</tbody>
</table>
Need for shared and exclusive locks

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>L(d)</td>
<td>L(d)</td>
</tr>
<tr>
<td></td>
<td>R(d)</td>
<td>R(d)</td>
</tr>
<tr>
<td>L(a)</td>
<td>W(a)</td>
<td>L(b)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R(b)</td>
</tr>
<tr>
<td>L(b)</td>
<td>W(b)</td>
<td></td>
</tr>
</tbody>
</table>

Precedence Graph

T1 → T2 → T3

It is conflict Serializable but requires granular control of locks.
Need for shared and exclusive locks

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>Lock requested</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SL(d) R(d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SL(b) SL(d)</td>
<td>R(b) R-SL(b)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XL(b) W(b) R-XL(b)</td>
<td>R(d) R-SL(d)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>XL(d) W(d)</td>
<td>R-XL(d)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lock held in mode</th>
<th>Lock requested</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Yes</td>
</tr>
<tr>
<td>X</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Reader/Writer (S/X)

• When accessing a DB Entity…
  • Table, Row, Column, Cell, etc…

• Before reading: Acquire a Shared (S) lock.
  • Any number of transactions can hold S.

• Before writing: Acquire an Exclusive (X) lock.
  • If a transaction holds an X, no other transaction
    can hold an S or X.