CS 764: Topics in Database Management Systems
Lecture 10: Isolation

Xiangyao Yu
10/10/2022
Announcement

Guest lecture on Wednesday (Oct. 12) from PingCAP (in-person)

Round-table discussion after the lecture
  – Time: 2:30—3:30 PM
  – Location: Room 4310 in CS department
A Critique of ANSI SQL Isolation Levels

Bill Branden 
Phil Berneman 
Jim Gray 
Jim Miliken 
Nancy Hall 
Patrick O'Neill

Abstract. ANSI SQL-92 [MS, ANSI] defines isolation levels in terms of transactions. Dirty Reads, Non-Repeatable Reads, and Phantom Reads and Phantoms. This paper shows that these phenomena and the ANSI SQL definitions fail to properly characterize several popular isolation levels, including the standard locking implementations of the levels covered. Ambiguity in the statement of the phenomena is inherent and a more formal statement is arrived at. Additionally, new phenomena that better characterize isolation types are introduced. Finally, an important nonsolvent isolation type, called Snapshot Isolation, is defined.

1. Introduction

Running concurrent transactions at different isolation levels allows application designers to trade off concurrency and through for transactions. Lower isolation levels increase transaction concurrency at the risk of allowing transactions that are inconsistent or out-of-date. Higher isolation levels ensure that concurrent transactions can execute at the highest isolation level (particular serializability) while concurrently existing transactions running at a lower isolation level can access data that are not yet committed or that possibly states of the transactions earlier (e.g., [3]). Of course, transactions running at lower isolation levels can produce invalid data. Application designers must guard against a later transaction running at a higher isolation level accessing invalid data and propagating such errors.

The ANSI SQL-92 specifications [MS, ANSI] define four isolation levels: (1) READ UNCOMMITTED, (2) READ COMMITTED, (3) REPEATABLE READ, and (4) SERIALIZABLE. These levels are defined with the classical availability by definition, (the three parallel operation subtypes, called occurrence, dirty read, non-repeatable read), and the serializability condition, called phantom reads and phantom reads, defined in the ANSI specifications, but the specifications suggest that phenomenon are essential subtypes that must (and cannot be assumed) (perhaps non-serializable) behavior.

We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what follows as ensuring that the operations of the subtypes for the level of the isolated data. We refer to serializability in what folle
Agenda

ANSI isolation levels
Cursor stability and snapshot isolation
Complexity of isolation
Agenda

**ANSI isolation levels**

Cursor stability and snapshot isolation

Complexity of isolation
Long vs. Short Locks

Short locks
- Locks held for the duration of a single action

Long locks
- Locks held to the end of the transaction

In strict two-phase locking, a transaction holds only long locks
Recap: Degree of Consistency

Degree 3: Serializability (assuming no phantom effect)
   – Long locks for reads and writes
Recap: Degree of Consistency

Degree 3: Serializability (assuming no phantom effect)
   – Long locks for reads and writes

Degree 2: Read Committed
   – Long locks for writes
   – Short locks for reads
Recap: Degree of Consistency

Degree 3: Serializability (assuming no phantom effect)
  – Long locks for reads and writes

Degree 2: Read Committed
  – Long locks for writes
  – Short locks for reads

Degree 1: Read Uncommitted
  – Long locks for writes
  – No lock for read
Recap: Degree of Consistency

Degree 3: Serializability (assuming no phantom effect)
  – Long locks for reads and writes

Degree 2: Read Committed
  – Long locks for writes
  – Short locks for reads

Degree 1: Read Uncommitted
  – Long locks for writes
  – No lock for read

Degree 0:
  – Short locks for writes
  – No lock for read
ANSI SQL-92 defines four isolation levels by phenomena
The original definitions were ambiguous

This lecture focuses on the “correct” definitions
Notation

\(w_1[x]\): transaction 1 writes record \(x\)

\(r_2[y]\): transaction 2 reads record \(y\)

\(w_1[P] \ (r_1[P])\): transaction 1 writes (reads) records that satisfy predicate \(P\)

\(c_1\): commit of transaction 1

\(a_1\): abort of transaction 1
Locking-Based Definition

**Well-formed**: lock (on tuple or predicate) before reading/writing records

**Long locks**: hold the lock until transaction commits or aborts

<table>
<thead>
<tr>
<th>Consistency Level = Locking Isolation Level</th>
<th>Read Locks on Data Items and Predicates (the same unless noted)</th>
<th>Write Locks on Data Items and Predicates (always the same)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree 3 = Locking SERIALIZABLE</td>
<td>Well-formed Reads, Long duration Read locks (both)</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
</tbody>
</table>
Locking-Based Definition

**Well-formed**: lock (on tuple or predicate) before reading/writing records

**Long locks**: hold the lock until transaction commits or aborts

<table>
<thead>
<tr>
<th>Consistency Level = Locking Isolation Level</th>
<th>Read Locks on Data Items and Predicates (the same unless noted)</th>
<th>Write Locks on Data Items and Predicates (always the same)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locking REPEATABLE READ</td>
<td>Well-formed Reads Long duration data-item Read locks</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td>Degree 3 = Locking SERIALIZABLE</td>
<td>Well-formed Reads Long duration Read locks (both)</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
</tbody>
</table>
Locking-Based Definition

**Well-formed**: lock (on tuple or predicate) before reading/writing records

**Long locks**: hold the lock until transaction commits or aborts

<table>
<thead>
<tr>
<th>Consistency Level = Locking Isolation Level</th>
<th>Read Locks on Data Items and Predicates (the same unless noted)</th>
<th>Write Locks on Data Items and Predicates (always the same)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locking REPEATABLE READ</td>
<td>Well-formed Reads, Long duration data-item Read locks</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td>Degree 3 = Locking SERIALIZABLE</td>
<td>Well-formed Reads, Short duration Read Predicate locks</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
</tbody>
</table>

**Phenomenon P3: Phantom**

r1[P]…w2[y in P]… (c1 or a1) and (c2 or a2) any order

- Anomalous behavior: multiple r[P]’s return different results

P3 is allowed in *repeatable read* but forbidden in *serializable*
Phantom Effect

Sailors

<table>
<thead>
<tr>
<th>Age</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>75</td>
<td>1</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
</tr>
<tr>
<td>85</td>
<td>2</td>
</tr>
</tbody>
</table>

T1: Find oldest sailors for ratings 1 and 2
T2: Insert (age:99, rating:1) and delete oldest sailor with rating 2
Phantom Effect

<table>
<thead>
<tr>
<th>Age</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>75</td>
<td>1</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
</tr>
<tr>
<td>85</td>
<td>2</td>
</tr>
</tbody>
</table>

T1: Find oldest sailors for ratings 1 and 2
T2: Insert (age:99, rating:1) and delete oldest sailor with rating 2

T1 locks oldest sailor in rating 1
Phantom Effect

Sailors

<table>
<thead>
<tr>
<th>Age</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>75</td>
<td>1</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
</tr>
<tr>
<td>85</td>
<td>2</td>
</tr>
<tr>
<td>99</td>
<td>1</td>
</tr>
</tbody>
</table>

T1: Find oldest sailors for ratings 1 and 2

T2: Insert (age:99, rating:1) and delete oldest sailor with rating 2

T1 locks oldest sailor in rating 1

T2 inserts a tuple with (age:99, rating:1)
Phantom Effect

T1: Find oldest sailors for ratings 1 and 2
T2: Insert (age:99, rating:1) and delete oldest sailor with rating 2

T1 locks oldest sailor in rating 1
T2 inserts a tuple with (age:99, rating:1)
T2 deletes oldest sailor with rating 2

<table>
<thead>
<tr>
<th>Age</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>75</td>
<td>1</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
</tr>
<tr>
<td>85</td>
<td>2</td>
</tr>
<tr>
<td>99</td>
<td>1</td>
</tr>
</tbody>
</table>
Phantom Effect

T1: Find oldest sailors for ratings 1 and 2
T2: Insert (age:99, rating:1) and delete oldest sailor with rating 2

T1 locks oldest sailor in rating 1
T2 inserts a tuple with (age:99, rating:1)
T2 deletes oldest sailor with rating 2
T2 commits
Phantom Effect

T1: Find oldest sailors for ratings 1 and 2
T2: Insert (age:99, rating:1) and delete oldest sailor with rating 2

T1 locks oldest sailor in rating 1
T2 inserts a tuple with (age:99, rating:1)
T2 deletes oldest sailor with rating 2
T2 commits
T1 locks oldest sailor in rating 2

<table>
<thead>
<tr>
<th>Age</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>75</td>
<td>1</td>
</tr>
<tr>
<td>85</td>
<td>2</td>
</tr>
<tr>
<td>99</td>
<td>1</td>
</tr>
</tbody>
</table>
Phantom Effect

<table>
<thead>
<tr>
<th>Sailors</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>Rating</td>
</tr>
<tr>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>75</td>
<td>1</td>
</tr>
<tr>
<td>85</td>
<td>2</td>
</tr>
<tr>
<td>99</td>
<td>1</td>
</tr>
</tbody>
</table>

T1: Find oldest sailors for ratings 1 and 2
T2: Insert (age:99, rating:1) and delete oldest sailor with rating 2

T1 locks oldest sailor in rating 1
T2 inserts a tuple with (age:99, rating:1)
T2 deletes oldest sailor with rating 2
T2 commits
T1 locks oldest sailor in rating 2
T1 commits. Output: (80,1), (85, 2)
Phantom Effect

T1: Find oldest sailors for ratings 1 and 2
T2: Insert (age:99, rating:1) and delete oldest sailor with rating 2

Output: (80, 1), (85, 2)

Different from all sequential execution output
- T1 -> T2. Output: (80, 1), (90, 2)
- T2 -> T1. Output: (99, 1), (85, 2)
Locking-Based Definition

**Well-formed**: lock (on tuple or predicate) before reading/writing records

**Long locks**: hold the lock until transaction commits or aborts

<table>
<thead>
<tr>
<th>Consistency Level</th>
<th>Read Locks on Data Items and Predicates (the same unless noted)</th>
<th>Write Locks on Data Items and Predicates (always the same)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree 2 = Locking READ COMMITTED</td>
<td>Well-formed Reads, Short duration Read locks (both)</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td>Locking REPEATABLE READ</td>
<td>Well-formed Reads, Long duration data-item Read locks, Short duration Read Predicate locks</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
</tbody>
</table>

**Phenomenon P2: Fuzzy Read**

r1[x]...w2[x]... (c1 or a1) and (c2 or a2) any order

- Anomalous behavior: multiple r[x]'s return different results

P2 is allowed in *read committed* but forbidden in *repeatable read*
Locking-Based Definition

**Well-formed:** lock (on tuple or predicate) before reading/writing records

**Long locks:** hold the lock until transaction commits or aborts

<table>
<thead>
<tr>
<th>Consistency Level = Locking Isolation Level</th>
<th>Read Locks on Data Items and Predicates (the same unless noted)</th>
<th>Write Locks on Data Items and Predicates (always the same)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree 1 = Locking READ UNCOMMITTED</td>
<td>none required</td>
<td>Well-formed Writes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Long duration Write locks</td>
</tr>
<tr>
<td>Degree 2 = Locking READ COMMITTED</td>
<td>Well-formed Reads</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td></td>
<td>Short duration Read locks (both)</td>
<td></td>
</tr>
</tbody>
</table>

**Phenomenon P1: Dirty Read**

- \( w_1[x] \ldots r_2[x] \ldots (c_1 \text{ or } a_1) \text{ and } (c_2 \text{ or } a_2) \) any order
- Anomalous behavior: transaction reads data that was never committed

P1 is allowed in *read uncommitted* but forbidden in *read committed*
Locking-Based Definition

**Well-formed**: lock (on tuple or predicate) before reading/writing records

**Long locks**: hold the lock until transaction commits or aborts

<table>
<thead>
<tr>
<th>Consistency Level = Locking Isolation Level</th>
<th>Read Locks on Data Items and Predicates (the same unless noted)</th>
<th>Write Locks on Data Items and Predicates (always the same)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree 0</td>
<td>none required</td>
<td>Well-formed Writes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Short duration Write locks</td>
</tr>
<tr>
<td>Degree 1 = Locking READ UNCOMMITTED</td>
<td>none required</td>
<td>Well-formed Writes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Long duration Write locks</td>
</tr>
</tbody>
</table>

**Phenomenon P0: Dirty Write**

\[w_1[x] \ldots w_2[x] \ldots (c_1 \text{ or } a_1) \text{ and } (c_2 \text{ or } a_2) \text{ any order}\]

- Anomalous behavior: when transaction 1 rolls back x, unclear what value to roll back to

P0 is forbidden in all ANSI isolation levels
## Equivalent Definitions

### Table 3. ANSI SQL Isolation Levels Defined in terms of the four phenomena

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>P0 Dirty Write</th>
<th>P1 Dirty Read</th>
<th>P2 Fuzzy Read</th>
<th>P3 Phantom</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ UNCOMMITTED</td>
<td>Not Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>READ COMMITTED</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>REPEATABLE READ</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>SERIALIZABLE</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Not Possible</td>
</tr>
</tbody>
</table>

### Consistency Level = Locking Isolation Level

<table>
<thead>
<tr>
<th>Consistency Level = Locking Isolation Level</th>
<th>Read Locks on Data Items and Predicates (the same unless noted)</th>
<th>Write Locks on Data Items and Predicates (always the same)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree 1 = Locking READ UNCOMMITTED</td>
<td>none required</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td>Degree 2 = Locking READ COMMITTED</td>
<td>Well-formed Reads, Short duration Read locks (both)</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td>Locking REPEATABLE READ</td>
<td>Well-formed Reads, Long duration data-item Read locks, Short duration Read Predicate locks</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td>Degree 3 = Locking SERIALIZABLE</td>
<td>Well-formed Reads, Short duration Read locks (both)</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
</tbody>
</table>
Hierarchy of Isolation Levels

Isolation level L1 is weaker than isolation level L2, denoted $L_1 \ll L_2$, if all non-serializable histories that obey the criteria of L2 also satisfy L1 and there is at least one non-serializable history that can occur at level L1 but not at level L2.

Read Uncommitted

$\ll$ Read Committed (RC)

$\ll$ Repeatable Read (RR)

$\ll$ Serializability (SR)
Agenda

ANSI isolation levels

Cursor stability and snapshot isolation

Complexity of isolation
### Cursor Stability

<table>
<thead>
<tr>
<th>Consistency Level = Locking Level</th>
<th>Read Locks on Data Items and Predicates (the same unless noted)</th>
<th>Write Locks on Data Items and Predicates (always the same)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Degree 2 = Locking READ COMMITTED</td>
<td>Well-formed Reads Short duration Read locks (both)</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td>Cursor Stability (see Section 4.1)</td>
<td>Well-formed Reads Read locks held on current of cursor Short duration Read Predicate locks</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td>Locking REPEATABLE READ</td>
<td>Well-formed Reads Long duration data-item Read locks Short duration Read Predicate locks</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
</tbody>
</table>

**Cursor**: can be viewed as a pointer to one row in a set of rows. The cursor can only reference one row at a time, but can move to other rows of the result set as needed.

**Phenomenon P4: Lost Update**

<table>
<thead>
<tr>
<th>r1[x]...w2[x]...w1[x]...c1</th>
</tr>
</thead>
<tbody>
<tr>
<td>– Anomalous behavior: transaction 2’s update is overwritten by transaction 1</td>
</tr>
</tbody>
</table>
All reads see a **snapshot** of data as of the time the transaction started (t1)

A transaction can commit if records in **write set** are not modified by other transactions between t1 and t2

At commit time, apply all writes with timestamp t2
Snapshot Isolation vs. Serializability

Anomaly A5B: Write Skew

r1[x]...r2[y]...w1[y]...w2[x]...(c1 or c2 occur)
- Transactions see a snapshot that does not reflect the latest updates
Anomaly A5B: Write Skew

\[ r_1[x] \ldots r_2[y] \ldots w_1[y] \ldots w_2[x] \ldots (c_1 \text{ or } c_2 \text{ occur}) \]
- Transactions see a snapshot that does not reflect the latest updates

In practice, snapshot isolation also requires the read snapshot reflects all the changes before the transaction starts
- Serializability requires no real-time ordering
- SI can be stronger than SR in this particular aspect
Snapshot Isolation vs. Serializability

Anomaly A5B: Write Skew

\[ r1[x]...r2[y]...w1[y]...w2[x]...(c1 \text{ or } c2 \text{ occur}) \]
- Transactions see a snapshot that does not reflect the latest updates

In practice, snapshot isolation also requires the read snapshot reflects all the changes before the transaction starts
- Serializability requires no real-time ordering
- SI can be stronger than SR in this particular aspect

Strict serializability (i.e., linearizability)
- Serializability + real-time constraint
- E.g., if transaction T1 commits before T2 starts, T1 must precede T2 in the serial order
Hierarchy of Isolation Levels

Serializable = Degree 3 = {Date, DB2} Repeatable Read

P3

P2

Repeatable Read

P2

Oracle

Consistent

Cursor Stability

P4C

P4C

Read Committed = Degree 2

P1

Read Uncommitted = Degree 1

P0

Degree 0

A5B

A5B

Snapshot

Isolation

A3

A3, A5A, P4
Agenda

ANSI isolation levels

Cursor stability and snapshot isolation

Complexity of isolation
Isolation is Complex

balance1 = 1000
balance2 = 1000

**constraint:**
balance1 + balance2 ≥ 1000

ball1 = read(balance1)
bal2 = read(balance1)
If ball1 + bal2 ≥ 2000
    ball1 = ball1 − 1000
    write(balance1, ball1)
    dispense cash
else
    reject
Isolation is Complex

\[ \text{balance1} = 1000 \]
\[ \text{balance2} = 1000 \]

**constraint:**
\[ \text{balance1} + \text{balance2} \geq 1000 \]

bal1 = read(balance1)
bal2 = read(balance1)
If bal1 + bal2 \geq 2000
   bal1 = bal1 - 1000
   write(balance1, bal1)
   dispense cash
else
   reject

bal1 = read(balance1)
bal2 = read(balance1)
If bal1 + bal2 \geq 2000
   bal2 = bal2 - 1000
   write(balance2, bal2)
   dispense cash
else
   reject
Isolation is Complex

balance1 = 1000
balance2 = 1000

**constraint:**

balance1 + balance2 ≥ 1000

---

1. bal1 = read(balance1)
   bal2 = read(balance1)
   
   If bal1 + bal2 ≥ 2000
   - bal1 = bal1 - 1000
   - write(balance1, bal1)
   - dispense cash
   
   else
   - reject

2. bal1 = read(balance1)
   bal2 = read(balance1)
   
   If bal1 + bal2 ≥ 2000
   - bal2 = bal2 - 1000
   - write(balance2, bal2)
   - dispense cash
   
   else
   - reject
Isolation is Complex

balance1 = 1000
balance2 = 1000

**constraint:**
balance1 + balance2 ≥ 1000

bal1 = read(balance1)
bal2 = read(balance1)
If bal1 + bal2 ≥ 2000
  bal1 = bal1 – 1000
  write(balance1, bal1)
  dispense cash
else
  reject

bal1 = read(balance1)
bal2 = read(balance1)
If bal1 + bal2 ≥ 2000
  bal2 = bal2 – 1000
  write(balance2, bal2)
  dispense cash
else
  reject
Isolation is Complex

balance1 = 1000
balance2 = 1000

constraint:
balance1 + balance2 ≥ 1000

ball1 = read(balance1)
ball2 = read(balance1)
If ball1 + ball2 ≥ 2000
  ball1 = ball1 − 1000
  write(balance1, ball1)
  dispense cash
else
  reject

ball1 = read(balance1)
ball2 = read(balance1)
If ball1 + ball2 ≥ 2000
  ball2 = ball2 − 1000
  write(balance2, ball2)
  dispense cash
else
  reject
Isolation is Complex

balance1 = 1000
balance2 = 1000

constraint:
balance1 + balance2 ≥ 1000

ball1 = read(balance1)
bal2 = read(balance1)
If ball1 + bal2 ≥ 2000

bal1 = bal1 - 1000
write(balance1, bal1)
dispense cash

else
reject

5

ball1 = read(balance1)
bal2 = read(balance1)
If ball1 + bal2 ≥ 2000

bal2 = bal2 - 1000
write(balance2, bal2)
dispense cash

else
reject

6

balance1 = 0 and balance2 = 0. **Constraint violated!**
ACID: Isolation – Why Strong Isolation?

MongoDB & Bitcoin: How NoSQL design flaws brought down two exchanges

DZone  April 2014

Attackers stole 896 Bitcoins ≈ 17 million US dollars
ACID: Isolation – Why Strong Isolation?

MongoDB & Bitcoin: How NoSQL design flaws brought down two exchanges

DZone  April 2014

Attackers stole 896 Bitcoins ≈ 17 million US dollars

Why you should pick strong consistency, whenever possible

Google Cloud   January 2018

“Systems that don't provide strong consistency … create a burden for application developers”
ACID: Isolation – Why Strong Isolation?

MongoDB & Bitcoin: How NoSQL design flaws brought down two exchanges

DZone   April 2014

Attackers stole 896 Bitcoins ≈ 17 million US dollars

Q: “What is the biggest mistake in your life as an engineer?”
A: (from Jeff Dean)

“Not putting distributed transactions in BigTable.

In retrospect lots of teams wanted that capability and built their own with different degrees of success.”  

March 2016
ACID: Isolation – Why Strong Isolation?

MongoDB & Bitcoin: How NoSQL design flaws brought down two exchanges

DZone April 2014

Attackers stole 896 Bitcoins ≈ 17 million US dollars

Q: “What is the biggest mistake in your life as an engineer?”
A: (from Jeff Dean) March 2016

Not putting distributed transactions in BigTable.
In retrospect lots of teams wanted that capability and built their own with different degrees of success.

SQL (before 2000) -> NoSQL (since 2000) -> NewSQL (since 2010s)
ACID: Isolation – Why Strong Isolation?

An alternative approach:
Optimize the performance of strong isolation instead of relaxing it

Q: “What is the biggest mistake in your life as an engineer?”
A: (from Jeff Dean)  

Not putting distributed transactions in BigTable.

In retrospect lots of teams wanted that capability and built their own with different degrees of success.

SQL (before 2000) -> NoSQL (since 2000) -> NewSQL (since 2010s)
Q/A – Isolation

How is snapshot isolation implemented nowadays?
Are these isolation levels used today?
Are there more isolation levels introduced in modern systems?
Do most applications need serializability?
Can multiple isolation levels coexist at transaction granularity?
What are desired properties of a good isolation level?
Next Lecture

Submit a review for the Wednesday guest lecture
  – Deadline: **Oct. 14, 11:59pm**
  – Use the same format as a paper review

Submit review by next Monday