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

Today's Paper: Isolation

A Critique of ANSI SQL Isolation Levels

Hal Berenson	Microsoft Corp.	haroldb@microsoft.com
Phil Bernstein	Microsoft Corp.	philbe@microsoft.com
Jim Gray	U.C. Berkeley	gray@crl.com
Jim Melton	Sybase Corp.	jim.melton@sybase.com
Elizabeth O'Neil	UMass/Boston	eoneil@cs.umb.edu
Patrick O'Neil	UMass/Boston	poneil@cs.umb.edu

Abstract: ANSI SQL-92 [MS, ANSI] defines Isolation *Levels* in terms of *phenomena:* Dirty Reads, Non-Repeatable 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 investigated and a more formal statement is arrived at; in addition new phenomena that better characterize isolation types are introduced. Finally, an important multiversion isolation type, called Snapshot Isolation, is defined.

1. Introduction

Running concurrent transactions at different isolation levels allows application designers to trade off concurrency and throughput for correctness. Lower isolation levels increase transaction concurrency at the risk of allowing transactions to observe a fuzzy or incorrect database state. Surprisingly, some transactions can execute at the highest isolation level (perfect serializability) while concurrently executing transactions running at a lower isolation level can access states that are not yet committed or that postdate states the transaction read earlier [GLPT]. 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 this invalid data and propagating such errors.

The ANSI/ISO SQL-92 specifications [MS, ANSI] define four isolation levels: (1) READ UNCOMMITTED, (2) READ COMMITTED, (3) REPEATABLE READ, (4) SERIALIZABLE. These levels are defined with the classical serializability definition, plus three prohibited operation subsequences, called *phenomena*: *Dirty Read*, *Non-repeatable Read*, and *Phantom*. The concept of a *phenomenon* is not explicitly defined in the ANSI specifications, but the specifications suggest that phenomena are operation subsequences that may lead to anomalous (perhaps non-serializable) behavior. We refer to *anomalies* in what follows when making suggested additions to the set of ANSI phenomena. As shown later, there is a technical distinction between anomalies and phenomena, but this distinction is not crucial for a general understanding.

The ANSI isolation levels are related to the behavior of lock schedulers. Some lock schedulers allow transactions to vary the scope and duration of their lock requests, thus departing from pure two-phase locking. This idea was introduced by [GLPT], which defined *Degrees of Consistency* in three ways: locking, data-flow graphs, and anomalies. Defining isolation levels by phenomena (anomalies) was intended to allow non-lock-based implementations of the SQL standard.

This paper shows a number of weaknesses in the anomaly approach to defining isolation levels. The three ANSI phenomena are ambiguous, and even in their loosest interpretations do not exclude some anomalous behavior that may arise in execution histories. This leads to some counter-intuitive results. In particular, lock-based isolation levels have different characteristics than their ANSI equivalents. This is disconcerting because commercial database systems typically use locking implementations. Additionally, the ANSI phenomena do not distinguish between a number of types of isolation level behavior that are popular in commercial systems. Additional phenomena to characterize these isolation levels are suggested here.

Section 2 introduces the basic terminology of isolation levels. It defines the ANSI SQL and locking isolation levels. Section 3 examines some drawbacks of the ANSI isolation levels and proposes a new phenomenon. Other popular isolation levels are also defined. The various definitions map between ANSI SQL isolation levels and the *degrees of consistency* defined in 1977 in [GLPT]. They also encompass Chris Date's definitions of Cursor Stability and Repeatable Read [DAT]. Discussing the isolation levels in a uniform framework reduces misunderstandings arising from independent terminology.

Section 4 introduces a multiversion concurrency control mechanism, called *Snapshot Isolation*, that avoids the ANSI SQL phenomena, but is not serializable. Snapshot Isolation is interesting in its own right, since it provides a reduced-isolation level approach that lies between READ COMMITTED and REPEATABLE READ. A new formalism (available in the longer version of this conference paper [OOBBGM]) connects reduced isolation levels for multiversioned data to the classical single-version locking serializability theory.

Section 5 explores some new anomalies to differentiate the isolation levels introduced in Sections 3 and 4. The extended ANSI SQL phenomena proposed here lack the power to characterize Snapshot isolation and Cursor Stability. Section 6 presents a Summary and Conclusions.

SIGMOD Record, 1995

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 Isolation Levels

	Table 1. ANSI SQL Isolation Levels Defined in terms of the Three Original Phenomena			
	Isolation Level	P1 (or A1) Dirty Read	P2 (or A2) Fuzzy Read	P3 (or A3) Phantom
Degree 1	ANSI READ UNCOMMITTED	Possible	Possible	Possible
Degree 2	ANSI READ COMMITTED	Not Possible	Possible	Possible
	ANSI REPEATABLE READ	Not Possible	Not Possible	Possible
Degree 3	ANOMALY SERIALIZABLE	Not Possible	Not Possible	Not Possible

ANSI SQL-92 defines four isolation levels by phenomena

The original definitions were ambiguous

This lecture focuses on the “correct” definitions

Notation

w1[x]: transaction 1 writes record x

r2[y]: transaction 2 reads record y

w1[P] (r1[P]): transaction 1 writes (reads) records that satisfy predicate P

c1: commit of transaction 1

a1: 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

Consistency Level = Locking Isolation Level	Read Locks on Data Items and Predicates (the same unless noted)	Write Locks on Data Items and Predicates (always the same)
Degree 3 = Locking SERIALIZABLE	Well-formed Reads Long duration Read locks (both)	Well-formed Writes, Long duration Write locks

Locking-Based Definition

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

Long locks: hold the lock until transaction commits or aborts

Consistency Level = Locking Isolation Level	Read Locks on Data Items and Predicates (the same unless noted)	Write Locks on Data Items and Predicates (always the same)
Locking REPEATABLE READ	Well-formed Reads Long duration data-item Read locks Short duration Read Predicate locks	Well-formed Writes, Long duration Write locks
Degree 3 = Locking SERIALIZABLE	Well-formed Reads Long duration Read locks (both)	Well-formed Writes, Long duration Write locks

Locking-Based Definition

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

Long locks: hold the lock until transaction commits or aborts

Consistency Level = Locking Isolation Level	Read Locks on Data Items and Predicates (the same unless noted)	Write Locks on Data Items and Predicates (always the same)
Locking REPEATABLE READ	Well-formed Reads Long duration data-item Read locks Short duration Read Predicate locks	Well-formed Writes, Long duration Write locks
Degree 3 = Locking SERIALIZABLE	Well-formed Reads Long duration Read locks (both)	Well-formed Writes, Long duration Write locks

Phenomenon P3: Phantom r1[P]

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

Age	Rating
80	1
75	1
90	2
85	2

T1: Find oldest sailors for ratings 1 and 2

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

Phantom Effect

Sailors

Age	Rating
80	1
75	1
90	2
85	2

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

Age	Rating
80	1
75	1
90	2
85	2
99	1

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

Sailors

Age	Rating
80	1
75	1
~~90~~	~~2~~
85	2
99	1

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

Phantom Effect

Sailors

Age	Rating
80	1
75	1
85	2
99	1

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

Sailors

Age	Rating
80	1
75	1
85	2
99	1

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

Phantom Effect

Sailors

Age	Rating
80	1
75	1
85	2
99	1

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

Sailors

Age	Rating
80	1
75	1
85	2
99	1

Phantom

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

Consistency Level = Locking Isolation Level	Read Locks on Data Items and Predicates (the same unless noted)	Write Locks on Data Items and Predicates (always the same)
Degree 2 = Locking READ COMMITTED	Well-formed Reads Short duration Read locks (both)	Well-formed Writes, Long duration Write locks
Locking REPEATABLE READ	Well-formed Reads Long duration data-item Read locks Short duration Read Predicate locks	Well-formed Writes, Long duration Write locks

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

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

Phenomenon P1: Dirty Read

w1[x]…r2[x]… (c1 or a1) and (c2 or a2) 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

Consistency Level = Locking Isolation Level	Read Locks on Data Items and Predicates (the same unless noted)	Write Locks on Data Items and Predicates (always the same)
Degree 0	none required	Well-formed Writes Short duration Write locks
Degree 1 = Locking READ UNCOMMITTED	none required	Well-formed Writes Long duration Write locks

Phenomenon P0: Dirty Write

w1[x]...w2[x]... (c1 or a1) and (c2 or a2) 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

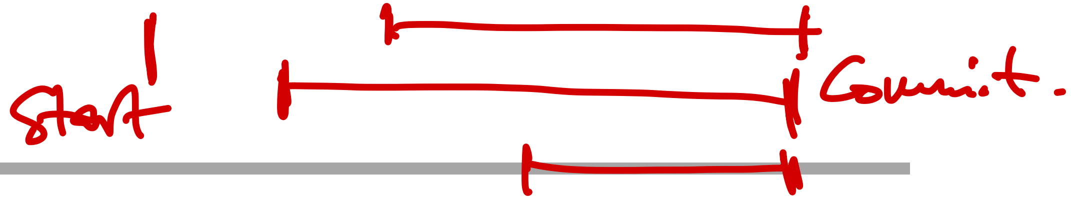


Table 3. ANSI SQL Isolation Levels Defined in terms of the four phenomena				
Isolation Level	**P 0 Dirty Write**	**P 1 Dirty Read**	**P 2 Fuzzy Read**	**P 3 Phantom**
READ UNCOMMITTED	Not Possible	Possible	Possible	Possible
READ COMMITTED	Not Possible	Not Possible	Possible	Possible
REPEATABLE READ	Not Possible	Not Possible	Not Possible	Possible
SERIALIZABLE	Not Possible	Not Possible	Not Possible	Not Possible

Consistency Level = Locking Isolation Level	**Read Locks on Data Items and Predicates (the same unless noted)**	**Write Locks on Data Items and Predicates (always the same)**
Degree 1 = Locking READ UNCOMMITTED	none required	Well-formed Writes Long duration Write locks
Degree 2 = Locking READ COMMITTED	Well-formed Reads Short duration Read locks (both)	Well-formed Writes, Long duration Write locks
Locking REPEATABLE READ	Well-formed Reads Long duration data-item Read locks Short duration Read Predicate locks	Well-formed Writes, Long duration Write locks
Degree 3 = Locking SERIALIZABLE	Well-formed Reads Long duration Read locks (both)	Well-formed Writes, Long duration Write locks

Hierarchy of Isolation Levels

Isolation level L1 is **weaker** than isolation level L2, denoted **L1 << L2**, 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

<< Read Committed (RC)

<< Repeatable Read (RR)

<< Serializability (SR)

SER

Agenda

ANSI isolation levels

Cursor stability and snapshot isolation

Complexity of isolation

Cursor Stability

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

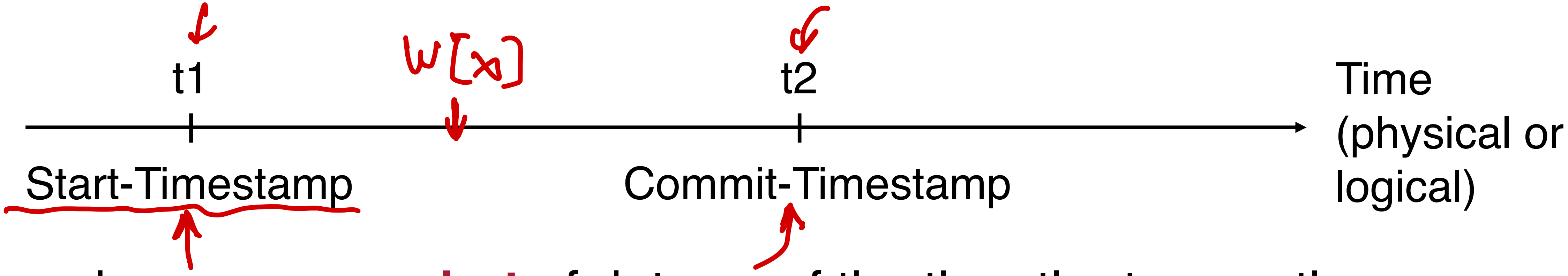
Cursor: can be viewed as a pointer to one row in a set of rows. The cursor can only reference one row at a time, but can move to other rows of the result set as needed

Phenomenon P4: Lost Update

r1[x]...w2[x]...w1[x]...c1

- Anomalous behavior: transaction 2's update is overwritten by transaction 1

Snapshot Isolation (SI)



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

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

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 or c2 occur)

- Transactions see a snapshot that does not reflect the latest updates

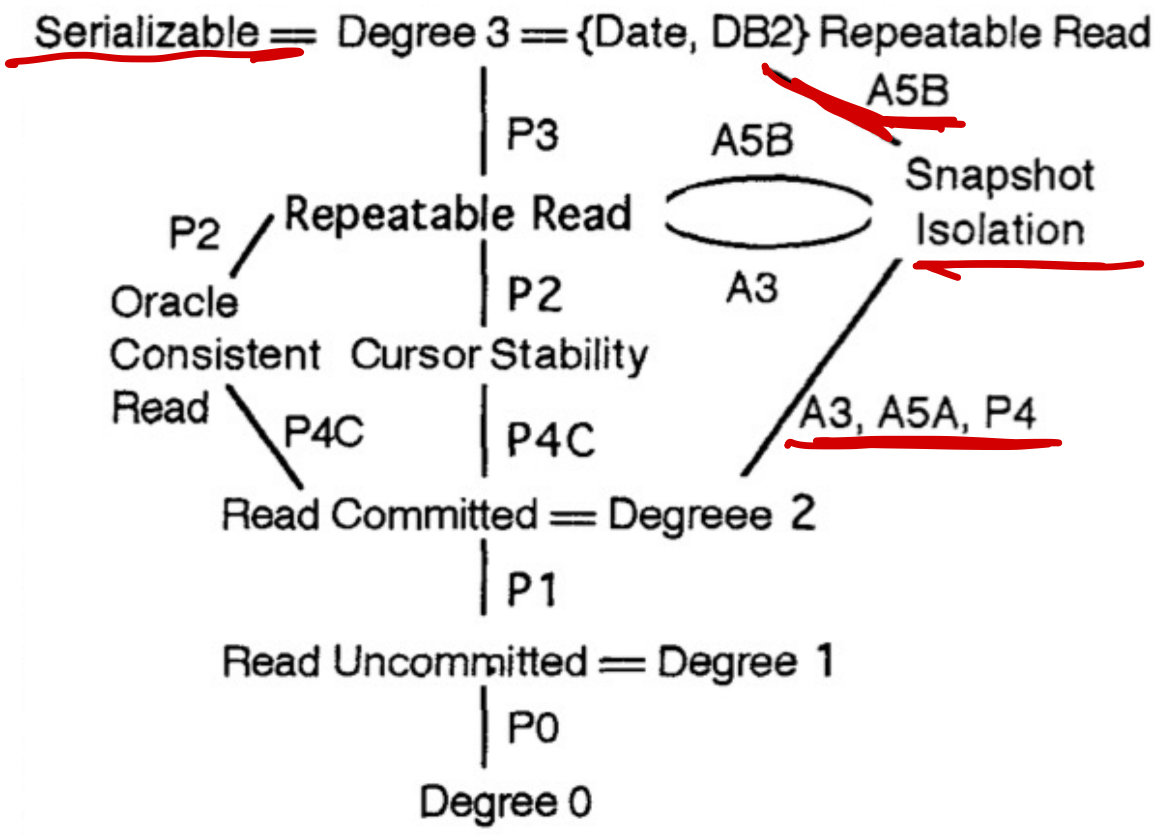
In practice, snapshot isolation also requires the read snapshot reflects all the changes before the transaction starts

- 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



Agenda

ANSI isolation levels

Cursor stability and snapshot isolation

Complexity of isolation

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
```

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

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
```

3

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

4

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
```

5

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

6

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
```

5

```
bal1 = read(balance1)
bal2 = read(balance1)
If bal1 + 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**) 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.

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**) 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)

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

- H. T. Kung, John T. Robinson, On Optimistic Methods for Concurrency Control. ACM Transactions on Database Systems, 1981