A Critique of ANSI SQL Isolation Levels

1. Introduction

Running concurrent transactions at different isolation levels allows application designers to trade off consistency and concurrency. However, isolation levels can suffer from transaction concurrency at the risk of allowing transactions to observe a theory or uncommitted database state. Surprisingly, some transactions can execute at the highest isolation level (serializable) while concurrently executing transactions running at a lower isolation level can access states that are not yet committed or that violate some of the transaction read rules [2]. 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 error.

The ANSI SQL-92 specifications [10, 11] define four isolation levels: (1) READ UNCOMMITTED, (2) READ COMMITTED, (3) REPEATABLE READ, and (4) SERIALIZABLE. These levels are defined with the classical serializability definitions: read-at-commit, write-at-date, and write-at-end definitions. However, the ANSI SQL-92 definitions are not explicitly defined in the ANSI specifications, but the specifications suggest that transactions are operation subsequence that may lead to anomalies (perhaps non serializable) behavior.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires specific permission and/or fee. ACM 1-86053-409-6/95/0005...$3.00

© 1995 ACM 0-89791-731-X/95/0005...$3.00

SIGMOD Record, 1995

Today's Paper: Isolation
Agenda

ANSI isolation levels
Cursor stability and snapshot isolation
Discussions
Recap: Degree of Consistency

Degree 3: Serializability (assuming no phantom effect)
   - Two-phase with respective to both reads and writes

Degree 2: Read Committed
   - Two-phase with respect to writes
   - Short read locks

Degree 1: Read Uncommitted
   - Two-phase with respect to writes
   - No read locks (may observe dirty data)

Degree 0:
   - Short write locks
   - No read locks
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>Short duration Read Predicate locks</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
</tbody>
</table>

**Phenomenon P3: Phantom**

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

P3 is allowed in *repeatable read* but forbidden in *serializable*
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 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>
<tr>
<td>Locking REPEATABLE READ</td>
<td>Well-formed Reads</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
<tr>
<td></td>
<td>Long duration data-item Read locks</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Short duration Read Predicate locks</td>
<td></td>
</tr>
</tbody>
</table>

**Phenomenon P2: Fuzzy Read**

- 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, 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>
</tbody>
</table>

**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

<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>
<thead>
<tr>
<th>Isolation Level</th>
<th>P 0 Dirty Write</th>
<th>P 1 Dirty Read</th>
<th>P 2 Fuzzy Read</th>
<th>P 3 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

<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 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, Long duration Read locks (both)</td>
<td>Well-formed Writes, Long duration Write locks</td>
</tr>
</tbody>
</table>
Discussion

Why define isolation levels? Why not serializability?

Why define isolation levels in this particular way?
Hierarchy of Isolation Levels

Isolation level L1 is \textit{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
    \( \ll \) Repeatable Read
      \( \ll \) Serializability
Cursor Stability

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

\[ r_1[x]...w_2[x]...w_1[x]...c_1 \]

- Anomalous behavior: transaction 2’s update is overwritten by transaction 1
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
Again, why define isolation levels in this particular way?
Isolation is Complex

Initially
checking.balance = 1000

bal = read(balance)
If bal > 100
   bal = bal - 100
   write(balance, bal)
   dispense cash
else
   reject
Isolation is Complex

Initially
checking.balance = 1000

bal = read(balance)
If bal > 100
  bal = bal – 100
  write(balance, bal)
  dispense cash
else
  reject

bal = read(balance)
If bal > 100
  bal = bal – 100
  write(balance, bal)
  dispense cash
else
  reject
Isolation is Complex

Initially
checking.balance = 1000

bal = read(balance)
If bal > 100
    bal = bal – 100
    write(balance, bal)
    dispense cash
else
    reject

bal = read(balance)
If bal > 100
    bal = bal – 100
    write(balance, bal)
    dispense cash
else
    reject
Isolation is Complex

Initially
checking.balance = 1000

bal = read(balance)
If bal > 100
  bal = bal – 100
  write(balance, bal)
  dispense cash
else
  reject

balance = 900

bal = read(balance)
If bal > 100
  bal = bal – 100
  write(balance, bal)
  dispense cash
else
  reject
ACID: Isolation – Why Strong Isolation?

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

DZone April 2014

Attackers stole 896 Bitcoins $\approx$ 46 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 ≈ 46 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 ≈ 46 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 ≈ 46 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)
Revisit Definition

**Serializability**: A transaction schedule is serializable if its outcome is equal to the outcome of its transactions executed serially.

<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>

Are both definitions above equivalent?
Revisit Definition

**Serializability**: A transaction schedule is serializable if its outcome is equal to the outcome of its transactions executed serially.

<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 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>

Are both definitions above equivalent?

Consider:
- T1: w1[x], r1[y], r1[z]
- T2: w2[x], r2[y], r2[z]
- Legal schedule? w1[x], w2[x], r1[y], r1[z], r2[y], r2[z], c1, c2
Optimize for Hotspots

Conventional 2PL

Bamboo [1]

Optimize for Hotspots

Conventional 2PL

T1
Update(A)

…

Commit

Wait for A

Update(A)

…

Commit

Update(A)

…

Commit

T2
Wait for A

Wait for A

Wait for A

T3
Wait for A

Wait for A

Wait for A

Bamboo [1]

Why not optimize performance for serializability instead of relaxing it?

[1] Zhihan Guo, et al. Releasing Locks As Early As You Can: Reducing Contention of Hotspots by Violating Two-Phase Locking, SIGMOD 2021
Q/A – Isolation

Isolation levels in commercial DBMS today?
  – Which are used? Are they all used?

The current ANSI standard?

Long running transactions in Snapshot Isolation?

Why do we need all these isolation levels?

Ongoing research to discover new phenomenon or isolation levels?
Before Next Lecture

Submit review for