Course Presentation
Distributed Database Systems

A Critique of ANSI SQL Isolation Levels
Microsoft Research
June 1995

Nikhil Wadhwa
Overview

- ANSI Specifications
- Phenomenon and Anomalies
- Broad v/s Strict Notations
- Isolation Levels
- Locking
- Cursor Stability
- Snapshot Isolation
- Conclusion
ANSI/ ISO-92 specifications

- READ UNCOMMITTED
- READ COMMITTED
- REPEATABLE READ
- SERIALIZABLE

- Above go from *weaker* to *stronger* isolation.
- Stronger isolation implies *less throughput* and *lesser anomalies*. 
Prohibited Action sequences or Phenomenon are action subsequences that may lead to *anomalous behavior*

Primary Types:

- Dirty Read
- Non-Repeatable Read
- Phantom
• A *transaction* groups a set of actions that transform the database from one consistent state to another.

• A *history* models the interleaved execution of a set of transactions as a linear ordering of their actions, such as Reads and Writes.

• Two actions in a history are said to *conflict* if they are performed by distinct transactions on the same data item and at least one of is a Write action.

• Conflicting actions can also occur on a set of data items, covered by a *predicate lock*, as well as on a single data item.

• A particular history gives rise to a *dependency graph* defining the temporal data flow among transactions.

• A history is serializable if it is equivalent to a *serial history* — that is, if it has the same dependency graph (inter-transaction temporal data flow) as some history that executes *transactions one at a time in sequence*. 
Phenomenon in Detail

• **P1 (Dirty Read):** Transaction T1 modifies a data item. Another transaction T2 then reads that data item before T1 performs a COMMIT or ROLLBACK. If T1 then performs a ROLLBACK, T2 has read a data item that was never committed and so never really existed.

• **P2 (Non-repeatable or Fuzzy Read):** Transaction T1 reads a data item. Another transaction T2 then modifies or deletes that data item and commits. If T1 then attempts to reread the data item, it receives a modified value or discovers that the data item has been deleted.

• **P3 (Phantom):** Transaction T1 reads a set of data items satisfying some <search condition>. Transaction T2 then creates data items that satisfy T1’s <search condition> and commits. If T1 then repeats its read with the same <search condition>, it gets a set of data items different from the first read.

• **Note:** None of these phenomena could occur in a serial history.

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>P1 (or A1) Dirty Read</th>
<th>P2 (or A2) Fuzzy Read</th>
<th>P3 (or A3) Phantom</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANSI READ UNCOMMITTED</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>ANSI READ COMMITTED</td>
<td>Not Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>ANSI REPEATABLE READ</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>ANOMALY SERIALIZABLE</td>
<td>Not Possible</td>
<td>Not Possible</td>
<td>Not Possible</td>
</tr>
</tbody>
</table>
One of the primary goals of the paper is to argue that the Anomaly (Strict) notation is not sufficient for real world cases, and the Phenomenon (Broad) notation is required.
• Each isolation level is characterized by the phenomena that a transaction is *forbidden* to experience (broad or strict interpretations).

• SERIALIZABLE isolation level must provide what is “commonly known as *fully serializable execution*.”

• It is a common misconception that *disallowing the three phenomena implies serializability.*

• Picking a broad interpretation of a phenomenon *excludes a larger* set of histories than the strict interpretation.

• Multiple versions of a data item may exist at one time in a *multi-version system*.
  • Any read must be *explicit* about which version is being read.
Transactions executing under a locking scheduler request Read (Share) and Write (Exclusive) locks on data items or sets of data items they read and write.

A Read (resp. Write) predicate lock on a given <search condition> is effectively a lock on all data items satisfying the <search condition>.

A transaction has well-formed writes (reads) if it requests a Write (Read) lock on each data item or predicate before writing (reading) that data item, or set of data items defined by a predicate.

A transaction has two-phase writes (reads) if it does not set a new Write (Read) lock on a data item after releasing a Write (Read) lock.

The locks requested by a transaction are of long duration if they are held until after the transaction commits or aborts.

Short locks are released immediately after the action completes.

Well-formed two-phase locking guarantees serializability.
Locking and Isolation Levels

<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>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>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>
<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>

- Isolation level L1 is *weaker* than isolation level L2 (or L2 is stronger than L1), denoted $L_1 \ « L_2$, 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.

- Two isolation levels are *incomparable*, denoted $L_1 \ »« L_2$, when each isolation level allows a non-serializable history that is disallowed by the other.

- Two isolation levels L1 and L2 are *equivalent*, denoted $L_1 == L_2$, if the sets of non-serializable histories satisfying L1 and L2 are identical.

- Locking READ UNCOMMITTED « Locking READ COMMITTED « Locking REPEATABLE READ « Locking SERIALIZABLE
Analysis of Isolation Levels

• Locking isolation levels are at least as isolated as the same-named ANSI levels.

• **P0 (Dirty Write):** Transaction T1 modifies a data item. Another transaction T2 then further modifies that data item before T1 performs a COMMIT or ROLLBACK. If T1 or T2 then performs a ROLLBACK, it is unclear what the correct data value should be.
  
  • **Broad interpretation: P0:** \( w_1[x]...w_2[x]...((c_1 \text{ or } a_1) \text{ and } (c_2 \text{ or } a_2) \text{ in any order}) \)

• Dirty writes can violate database consistency.
  
  • \( w_1[x] w_2[x] w_2[y] c_2 w_1[y] c_1, \text{ with the constraint } x = y \)

• ANSI SQL isolation should be modified to require P0 for all isolation levels.

• Balance transfer case: $40 transfer between bank balance rows x and y

  • \( H_1: r_1[x=50]w_1[x=10]r_2[x=10]r_2[y=50]c_2 r_1[y=50]w_1[y=90]c_1 \)

  • T2 reads an inconsistent state with a total balance of 60, it should be 100

  • \( H_1 \) does not violate A1, A2, A3. \( H_1 \) violates P1.

  • Hence it is argued that the broad (Phenomenon) interpretation is the correct one, not anomalous one

• For single version histories, it turns out that the P0, P1, P2, P3 phenomena are disguised versions of locking.
• Designed to prevent the *lost update phenomenon*.

• **P4 (Lost Update)**: The lost update anomaly occurs when transaction T1 reads a data item and then T2 updates the data item (possibly based on a previous read), then T1 (based on its earlier read value) updates the data item and commits. In terms of histories, this is:
  
  • P4: $r_1[x]...w_2[x]...w_1[x]...c_1$

• Extends READ COMMITTED locking behavior for SQL cursors by adding a new read action for FETCH from a cursor and *requiring that a lock be held* on the current item of the cursor.

• The lock is held until the cursor moves or is *closed*, possibly by a commit.
Each transaction reads data from a snapshot of the (committed) data as of the time the transaction started, called its **Start-Timestamp**.

The transaction's writes (updates, inserts, and deletes) will be reflected in this snapshot, to be read again if the transaction accesses (i.e., reads or updates) the data a second time.

Updates by other transactions active after the transaction Start-Timestamp are **invisible** to the transaction.

Snapshot Isolation is a type of **multiversion** concurrency control.

When the transaction T1 is ready to commit, it gets a **Commit-Timestamp**, which is larger than any existing Start-Timestamp or Commit-Timestamp.

The transaction **successfully commits only if** no other transaction T2 with a Commit-Timestamp in T1’s execution interval [Start-Timestamp, Commit-Timestamp] wrote data that T1 also wrote.

This feature, called **First-committer-wins** prevents lost updates (P4).

40$ transfer case:

- H1.SI: r1[x0=50] w1[x1=10] r2[x0=50] r2[y0=50] c2 r1[y0=50] w1[y1=90] c1
A5A - Read Skew: Suppose transaction T1 reads x, and then a second transaction T2 updates x and y to new values and commits. If now T1 reads y, it may see an inconsistent state, and therefore produce an inconsistent state as output. In terms of histories, we have the anomaly:

A5A: r1[x]...w2[x]...w2[y]...c2...r1[y]...(c1 or a1)

A5B - Write Skew: Suppose T1 reads x and y, which are consistent with C(), and then a T2 reads x and y, writes x, and commits. Then T1 writes y.

A5B: r1[x]...r2[y]...w1[y]...w2[x]...(c1 and c2 occur)
Conclusion

- There are *serious problems* with the original ANSI SQL definition of isolation levels
  - Dirty Writes (P0) are *not precluded*.
- Many applications avoid lock contention by using *Cursor Stability* or Oracle's Read Consistency isolation.
Thank You!

Paper: A Critique of ANSI SQL Isolation Levels (June 1995)