A Critique Of Ansi SQL Isolation Levels

March 18, 2025

1 Introduction

The ANSI/ISO SQL-92 specifications 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 action subsequences, called phenomena: Dirty Read, Non-repeatable Read, and Phantom.

This paper shows a number of weaknesses in the anomaly approach to defining isolation levels. The three ANSI phenomena are ambiguous.

2 Isolation Definitions

2.1 Serializability Concepts

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 (i.e., inserts, updates, and deletes) of specific data items. 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.

A particular history gives rise to a **dependency graph** defining the temporal data flow among transactions. The actions of committed transactions

in the history are represented as graph nodes. If action op_1 of transaction T_1 conflicts with and precedes action op_2 of transaction T_2 in the history, then the pair $\langle op_1, op_2 \rangle$ becomes an edge in the dependency graph. Two histories are **equivalent** if they have the same committed transactions and the same dependency graph. 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.

2.2 ANSI SQL Isolation Levels

- P1 (Dirty Read): Transaction T_1 modifies a data item. Another transaction T_2 then reads that data item before T_1 performs a `COMMIT` or `ROLLBACK`. If T_1 then performs a `ROLLBACK`, T_2 has read a data item that was never committed and so never really existed.

- P2 (Non-repeatable or Fuzzy Read): Transaction T_1 reads a data item. Another transaction T_2 then modifies or deletes that data item and commits. If T_1 then attempts to reread the data item, it receives a modified value or discovers that the data item has been deleted.

- P3 (Phantom): Transaction T_1 reads a set of data items satisfying some `<search condition>`. Transaction T_2 then creates data items that satisfy T_1's `<search condition>` and commits. If T_1 then repeats its read with the same `<search condition>`, it gets a set of data items different from the first read.

Histories consisting of reads, writes, commits, and aborts can be written in a shorthand notation: $w_1[x]$ means a write by transaction 1 on data item x , and $r_2[x]$ represents a read of x by transaction 2. Transaction 1 reading and writing a set of records satisfying predicate P is denoted by $r_1[P]$ and $w_1[P]$ respectively. Transaction 1's commit and abort (`ROLLBACK`) are written c_1 and a_1, respectively.

P1 might be restated as disallowing the following scenario:

$$w_1[x] \ldots r_2[x] \ldots (a_1 \text{ and } c_2 \text{ in any order}) \tag{1}$$

But P1 does not insist that T_1; it simply states that if this happens something might occur. Therefoe we can also interpret P1 as:

$$w_1[x] \ldots r_2[x] \ldots ((c_1 \text{ or } a_1) \text{ and } (c_2 \text{ or } a_2) \text{ in any order}) \tag{2}$$

Forbidding (2) variant of P1 disallows any history where T_1 modifes a data item x, then T_2 reads the data item before T_1 commits or aborts. It does not insist that T_1 aborts or that T_2 commits.

We call (1) the **strict interpretation** of P1 and (2) the **broad interpretation** of P1. Interpretation (2) specifies a phenomenon that might lead to an anomaly, while (1) specifies an actual anomaly. Denote them as P1 and A1 respectively.

Similarly, the English language phenomena P2 and P3 have strict and broad interpretations, and are denoeted as:

- P2: $r_1[x] \dots w_2[x] \dots ((c_1 \text{ or } a_1) \text{ and } (c_2 \text{ or } a_2) \text{ in any order})$
- A2: $r_1[x] \dots w_2[x] \dots c_2 \dots r_1[x] \dots c_1$
- P3: $r_1[P] \dots w_2[y \in P] \dots ((c_1 \text{ or } a_1) \text{ and } (c_2 \text{ or } a_2) \text{ in any order})$
- A3: $r_1[P] \dots w_2[y \in P] \dots c_2 \dots r_1[P] \dots c_1$

ANSI SQL defines four levels of isolation by the matrix of Table 1

Isolation Level	P1	P2	P3
ANSI READ UNCOMMITTED	Possible	Possible	Possible
ANSI READ COMMITTED	Not Possible	Possible	Possible
ANSI REPEATABLE READ	Not Possible	Not Possible	Possible
ANOMALY SERIALIZABLE	Not Possible	Not Possible	Not Possible

2.3 Locking

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. Two locks by different transactions on the same item **conflict** if at least one of them is a Write lock.

A Read (resp. Write) predicate lock on a given `<search condition>` is effectively a lock on all data items satisfying the `<search condition>`. This may be an infinite set. It includes data present in the database and also any phantom data items not currently in the database but that would satisfy the predicate if they were inserted or if current data items were updated to satisfy the `<search condition>`. In SQL terms, a predicate lock covers all-tuples that satisfy the predicate and any that an `INSERT`, `UPDATE`, or `DELETE` statement would cause to satisfy the predicate. Two predicate locks by different transactions conflict if one is a Write lock and if there is a (possibly

phantom) data item covered by both locks. An item lock (record lock) is a predicate lock where the predicate names the specific record.

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. The transaction is **well-formed** if it has well-formed writes and reads. 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. A transaction exhibits **two-phase locking** if it does not request any new locks after releasing some lock.

The locks requested by a transaction are of **long duration** if they are held until after the transaction commits or aborts. Otherwise, they are of **short duration**. Typically, short locks are released immediately after the action completes.

The fundamental serialization theorem is that well-formed two-phase locking guarantees serializability — each history arising under two-phase locking is equivalent to some serial history. Conversely, if a transaction is not well-formed or two-phased then, except in degenerate cases, non-serializable execution histories are possible

Table 2.3 defines a number of isolation types in terms of lock scopes (items or predicates), modes (read or write), and their durations (short or long). We believe the isolation levels called Locking READ UNCOMMITTED, Locking READ COMMITTED, Locking REPEATABLE READ, and Locking SERIALIZABLE are the locking definitions intended by ANSI SQL Isolation levels

Table 2. Degrees of Consistency and Locking Isolation Levels defined in terms of locks.

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
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
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
Degree 3 = Locking SERIALIZABLE	Well-formed Reads Long duration Read locks (both)	Well-formed Writes, Long duration Write locks

Definition 2.1. Isolation level L_1 is **weaker** than isolation level L_2 (or L_2 is stronger than L_1), 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 L_1 and L_2 are **equivalent**, denoted $L_1 = L_2$, if the sets of non-serializable histories satisfying L_1 and L_2 are identical. L_1 is **no stronger** than L_2, denoted

$L_1 \leq L_2$ if either $L_1 < L_2$ or $L_1 = L_2$. 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.

In comparing isolation levels we differentiate them only in terms of the non-serializable histories that can occur one but not the other. Two isolation levels can also differ in terms of the serializable histories they permit, but we say Locking SERIALIZABLE $=$ Serializable even though it is well known that a locking scheduler does not admit all possible Serializable histories.

$$\begin{aligned} \text{Locking READ COMMITTED} &< \text{Locking READ COMMITTED} \\ &< \text{Locking REPEATABLE READ} \\ &< \text{Locking SERIALIZABLE} \end{aligned}$$

3 Analyzing ANSI SQL Isolation Levels

Locking READ UNCOMMITTED provides long duration write locking to avoid a phenomenon called "Dirty Writes," but ANSI SQL does not exclude this anomalous behavior other than ANSI SERIALIZABLE. Dirty writes are defined as follows:

- P0 (Dirty Write): Transaction T_1 modifies a data item. Another transaction T_2 then further modifies that data item before T_1 performs a `COMMIT` or `ROLLBACK`. If T_1 or T_2 then performs a `ROLLBACK`, it is unclear whtat the correct data value should be

$$\text{P0: } w_1[x] \ldots w_2[x] \ldots ((c_1 \text{ or } a_1) \text{ and } (c_2 \text{ or } a_2) \text{ in any order})$$

Without protection from P0, the system can't undo updates by restoring before images. Consider the history: $w_1[x]w_2[x]a_1$. You don't want to undo $w_1[x]$ by restoring its before-image of x, because that would wipe out w_2's update. But if you don't restore its before-image, and transaction T_2 later aborts, you can't undo $w_2[x]$ by restoring its before-image either!

So we conclude:

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

Consider history H_1:

$$H_1 : r_1[x = 50]w_1[x = 10]r_2[x = 10]r_2[y = 50]c_2r_1[y = 50]w_1[y = 90]c_1$$

H_1 is non-serializable, the classical inconsistent analysis problem where transaction T_1 is transferring a quantity 40 from x to y, maintaining a total balance of 100, but T_2 reads an inconsistent state where the total balance is 60.

But H_1 does not violate any of the anomalies A_1, A_2 or A_3. But consider instead taking the broad interpretation of A_1, the phenomenon P_1:

$$w_1[x] \dots r_2[x] \dots ((c_1 \text{ or } a_1) \text{ and } (c_2 \text{ or } a_2) \text{ in any order})$$

H_1indeed violates P_1.

Similar arguments show that P_2 should be taken as the ANSI intention rather than A_2. A history that discriminates these two interpretations is:

$$H_2 : r_1[x = 50] r_2[x = 50] w_2[x = 10] r_2[y = 50] w_2[y = 90] c_2 r_1[y = 90] c_1$$

H_2 is non-serializable, where T_1 sees a total balance of 140. This time neither transaction reads dirty data. Thus P_1 is satisfied. Once again, no data item is read twice nor is any relevant predicate evaluation changed. The problem with H_2 is that by the time T_1 reads y, the value for x is out of date. If T_1 were to read x again, it would have been changed; but since T_1 doesn't do that, A_2 doesn't apply.

In essence, T_1 doesn't aware of T_2's existence and can't determine if its read value is the newest.

Finally, consider

$$H_3 : r_1[P] w_2[\text{insert } y \in P] r_2[z] w_2[z] c_2 r_1[z] c_1$$

Here T_1 performs a `<search condition>` to find the list of active employees. Then T_2 performs an insert of a new active employee and then updates z, the count of employees in the company. Following this, T_1 reads the count of active employees as a check and sees a discrepancy. This history is clearly not serializable, but is allowed by A_3 since no predicate is evaluated twice.

Remark. Strict interpretations A_1, A_2 and A_3 have unintended weakness. The correct interpretations are the Broad ones. We assume in what follows that ANSI meant to define P_1, P_2 and P_3

Remark.

- P0 (Dirty Write): $w_1[x] \dots w_2[x] \dots (c_1 \text{ or } a_1)$
- P1 (Dirty Read): $w_1[x] \dots r_2[x] \dots (c_1 \text{ or } a_1)$
- P2 (Fuzzy or Non-repeatable Read): $r_1[x] \dots w_2[x] \dots (c_1 \text{ or } a_1)$

- P3 (Phantom): $r_1[P] ... w_2[y \in P] ... (c_1 \text{ or } a_1)$

The definition of proposed ANSI isolation levels in terms of these phenomena is given in Table 3.

For single version histories, it turns out that the P0, P1, P2, P3 phenomena are disguised versions of locking.

- prohibiting P0 precludes a second transaction writing an item after the first transaction has written it, equivalent to saying that long-term Write locks are held on data items (and predicates). Thus Dirty Writes are impossible at all levels.
- Similarly, prohibiting P1 is equivalent to having well-formed reads on data items.
- Prohibiting P2 means long-term Read locks on data items.
- Finally, Prohibiting P3 means long-term Read predicate locks.

Thus the isolation levels of Table 3 defined by these phenomena provide the same behavior as the Locking isolation levels of Table 2.3.

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

Isolation Level	P0 Dirty Write	P1 Dirty Read	P2 Fuzzy Read	P3 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

Remark. The locking isolation levels of Table 2.3 and the phenomenological definitions of Table 3 are equivalent. Put another way, P0, P1, P2, and P3 are disguised redefinition's of locking behavior.

4 Other Isolation Types

4.1 Cursor Stability

P4 (Lost Update): The lost update anomaly occurs when transaction T_1 reads a data item and then T_2 updates the data item (possibly based on a previous read), then T_1 (based on its earlier read value) updates the data item and commits.

$$\text{P4 (Lost Update): } r_1[x] ... w_2[x] ... w_1[x] ... c_1$$

Remark. READ COMMITTED $<$ Cursor Stability $<$ REPEATABLE READ

4.2 Snapshot Isolation

Snapshot Isolation: each transaction read reads data from a snapshot of the (committed) data as of the time the transaction started, called its Start-Timestamp.

When the transaction T_1 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 T_2 with a Commit-Timestamp in T_1's execution interval [Start-Timestamp, Commit-Timestamp] wrote data that T1 also wrote. Otherwise, T1 will abort. This feature, called **First-committer-wins** prevents lost updates. When T_1 commits, its changes become visible to all transactions whose Start-Timestamps are larger than T1's Commit-Timestamp.

Snapshot Isolation is non-serializable because a transaction's Reads come at one instant and the Writes at another. For example, consider the single-value history

$$\text{H5:} r_1[x=50] r_1[y=50] r_2[x=50] r_2[y=50] w_1[y=-40] w_2[x=-40] c_1 c_2$$

Here we assume that each transaction that writes a new value for x and y is expected to maintain the constraint that x + y should be positive, and while T1 and T2 both act properly in isolation, the constraint fails to hold in H5.

A5 (Data Item Constraint Violation). Suppose $C()$ is a database constraint between two data items x and y in the database. Here are two anomalies arising from constraint violation.

A5A Read Skew: Suppose transaction T_1 reads x, and then a second transaction T_2 updates x and y to new values and commits. If now T_1 reads y, it may see an inconsistent state, and therefore produce an inconsistent state as output.

$$\text{A5A (Read Skew): } r_1[x] \dots w_2[x] \dots w_2[y] \dots c_2 \dots r_1[y] (c_1 \text{ or } a_1)$$

A5B Write Skew: Suppose T_1 reads x and y, which are consistent with $C()$, and then a T_2 reads x and y, writes x, and commits. Then T_1 writes y. If there were a constraint between x and y, it might be violated.

$$\text{A5B (Write Skew): } r_1[x] \dots r_2[y] \dots w_1[y] \dots w_2[x] \dots (c_1 \text{ and } c_2 \text{ occur})$$

Clearly neither A5A nor A5B could arise in histories where P2 is precluded, since both A5A and A5B have T2 write a data item that has been previously read by an uncommitted T1. Thus, phenomena A5A and A5B are only useful for distinguishing isolation levels that are below REPEATABLE READ in strength.

Proposition 4.1. *READ COMMITTED $<$ Snapshot Isolation*

Proof. In Snapshot Isolation, first-committer-wins precludes P0 (dirty writes), and the timestamp mechanism prevents P1 (dirty reads), so Snapshot Isolation is no weaker than READ COMMITTED. In addition, A5A is possible under READ COMMITTED, but not under the Snapshot Isolation timestamp mechanism. Therefore READ COMMITTED $<$ Snapshot Isolation □

Proposition 4.2. *REPEATABLE READ $><$ Snapshot Isolation*

Proof. Snapshot Isolation histories prohibit histories with anomaly A3, but allow A5B, while REPEATABLE READ does the opposite □

Remark. Snapshot Isolation histories preclude anomalies A1, A2 and A3. Therefore

$$\text{ANOMALY SERIALIZABLE} < \text{Snapshot Isolation}$$

5 Summary

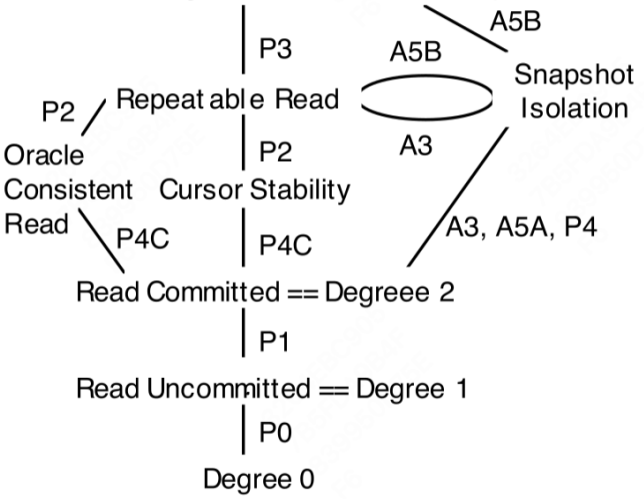

Figure 2: A diagram of the isolation levels and their relationships. It assume that the ANSI SQL isolation levels have been strengthened to match the recommendation of Remark 5 and Table 3. The edges are annotated with the phenomena that differentiate the isolation levels. Not shown is a potential multi-version hierarchy extending Snapshot Isolation to lower degrees of isolation by picking read timestamps on a per-statement basis. Nor does it show the original ANSI SQL isolation levels based on the strict interpretation of the phenomenon P1, P2, and P3.

6 Problems

7 References