

Transaction Isolation in Mixed-Level and Mixed-Scope Settings

Stephen J. Hegner

Retired from:

Umeå University, Sweden

<http://people.cs.umu.se/hegner>

Currently:

DBMS Research of New Hampshire, USA

dbmsnh@gmx.com

Often visiting:

University of Concepción, Chile

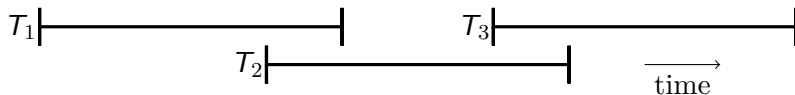
Expanded from ADBIS 2019 presentation

Isolation of Database Transactions

Transactions: A database transaction performs reads and (possibly) writes on a database.

Examples on banking database: Read account balance, withdraw amount x from account a , transfer amount x from account a_1 to account a_2 .

Concurrency: In a modern DBMS, transactions may run *concurrently*.



Isolation: Concurrent transactions must not interfere with one another.

Lost Update — Result of Limited Isolation

- If two concurrent transactions read and write the same data object, a *lost update* may result.

time ↓

Global DB x	T_1		T_2	
	Operation	Local x	Operation	Local x
10000	Read $\langle x \rangle$	10000		
10000		10000	Read $\langle x \rangle$	10000
10000	Compound $\langle x, 10\% \rangle$	11000		10000
10000		11000	Withdraw $\langle x, 2000 \rangle$	8000
11000	Write $\langle x \rangle$	11000		8000
8000		11000	Write $\langle x \rangle$	8000

- The update of T_1 is lost completely; only the update of T_2 is retained in the global DB.
- This behavior can happen in real systems (e.g., PostgreSQL) with transaction isolation level READ COMMITTED.
- Lost update is only one example of an *update anomaly*; there are many others.

Write Skew — Result of Limited Isolation

- Imperfect isolation may occur even in the case that two transactions write different objects.
- Think of x and y as (the balances of) bank accounts with the constraint that $x + y \geq 500$.
- The following illustrates *write skew*.

Global DB (x, y)	T_1		T_2	
	Operation	Local (x, y)	Operation	Local (x, y)
(300, 300)	Read(x)	(300, -)		
(300, 300)	Read(y)	(300, 300)		
(300, 300)	If $x+y \geq 600$ then Withdraw(x, 100)	(200, 300)		
(300, 300)		(200, 300)	Read(y)	(-, 300)
(300, 300)		(200, 300)	Read(x)	(300, 300)
(300, 300)		(200, 300)	If $x+y \geq 600$ then Withdraw(y, 100)	(300, 200)
(200, 300)	Write(x)	(200, 300)		(300, 200)
(200, 200)		(200, 300)	Write(y)	(300, 200)

time ↓

Consistency: T_1 and T_2 are each *consistent*; they respect the integrity constraint when run individually.

Levels of Isolation

Schedule: A finite set of transactions, each with start time, end time, and times for the read and write operations, is a *schedule*.

Serializable schedule: The behavior of the (possibly concurrent) transactions must be equivalent to that for some non-concurrent, or *serial* schedule.

- To obtain a serial schedule, the transactions are “shifted” in time, but otherwise unchanged.

Gold standard for isolation: Serializable schedule.

Limited concurrency: Serializable schedules may limit concurrency (and hence performance) substantially.

Reality: DBMSs offer a variety of isolation levels.

Tradeoff:

- Higher isolation \Rightarrow reduced concurrency.
- Lower isolation \Rightarrow undesirable interaction.

SQL: ~~READ UNCOMMITTED~~

< READ COMMITTED < REPEATABLE READ < SERIALIZABLE.

Goals of this Research

- In descriptions of the various isolation levels, it is typically assumed that all transactions run at the same level.

Reality: Most DBMSs permit the selection of isolation level on a per-transaction basis.

Goal 1 – mixed-level isolation: Develop a systematic model of the behavior of transactions when different ones run at different levels of isolation.

Example: T_1 runs under REPEATABLE READ while while T_2 runs under READ COMMITTED.

Local scope: READ COMMITTED and REPEATABLE READ are *local* in scope.

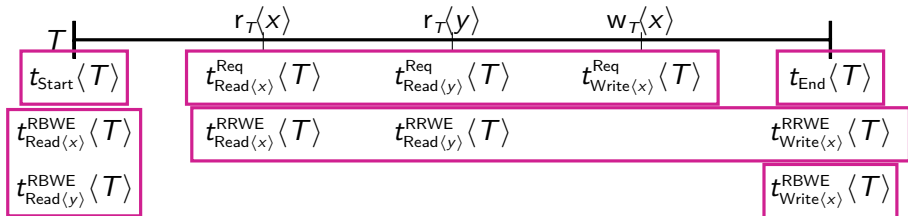
- They are properties of an individual transaction, depending only upon that transaction and its relationship to concurrent neighbors.

Global scope: SERIALIZABLE is a property of a *schedule* of transactions.

- It makes no sense to say that an individual transaction is serializable.

Goal 2 – mixed-scope isolation: Develop a model in which SERIALIZABLE isolation has meaning in a mixed-level setting.

The Object-Level Model of Transactions



- Each transaction T has a *start time* and an *end time*.
- Read and write operations are at the *object level*.
 - Operations, but not the values, are modelled.
- Each read and each write operation has a *request time*.
- In a *Teff-transaction* $\langle T, \tau \rangle$, each read and write operation also has an *effective time assignment* τ , at which the global DBMS is read or written.
 - $\tau = RRWE = \textit{read (at) request write (at) end}$.
 - $\tau = RBWE = \textit{read (at) beginning write (at) end (snapshot read)}$.

The Conflict Graph (DSG)

Context: $S = \{\langle T_i, \tau_i \rangle \mid 1 \leq i \leq k\}$ Teff-transactions.

Time compatible: Distinct transactions have no time points in common.

Time points: $t_{\text{Start}}\langle T_i \rangle$, $t_{\text{End}}\langle T_i \rangle$, $t_{\text{Read}\langle T_i \rangle}^{\tau_i}\langle x \rangle$, $t_{\text{Read}\langle T_i \rangle}^{\text{Req}}\langle x \rangle$, $t_{\text{Write}\langle T_i \rangle}^{\tau_i}\langle x \rangle$,
 $t_{\text{Write}\langle T_i \rangle}^{\text{Req}}\langle x \rangle$.

Direct Serialization Graph (DSG): Has S as vertices and edges as follows:

$\langle T_i, \tau_i \rangle \xrightarrow{rw} \langle T_j, \tau_j \rangle$: T_i reads some x and T_j is the next writer of x .

$\langle T_i, \tau_i \rangle \xrightarrow{ww} \langle T_j, \tau_j \rangle$: T_i writes some x and T_j is the next writer of x .

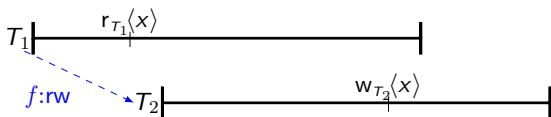
$\langle T_i, \tau_i \rangle \xrightarrow{wr} \langle T_j, \tau_j \rangle$: T_i is the last writer of some x before T_j reads x .

- *Effective* times are used in all three types of conflict:

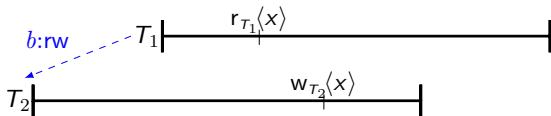
Edge Sense in the DSG

- The *sense* of an edge is very important in this work.

Forward edge: $\langle T_1, \tau_1 \rangle \xrightarrow{f:zz} \langle T_2, \tau_2 \rangle$ holds iff $t_{\text{End}}\langle T_1 \rangle < t_{\text{End}}\langle T_2 \rangle$.



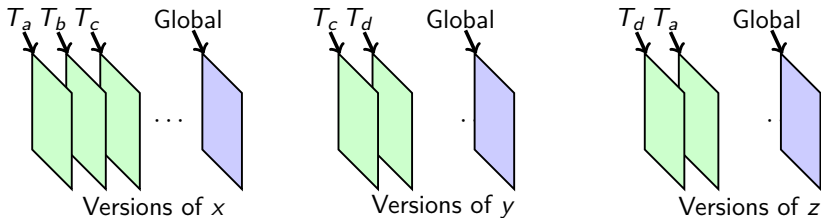
Backward edge: $\langle T_1, \tau_1 \rangle \xrightarrow{b:zz} \langle T_2, \tau_2 \rangle$ holds iff $t_{\text{End}}\langle T_2 \rangle < t_{\text{End}}\langle T_1 \rangle$.



Fact: Only rw-edges can be backward; ww- and wr-edges must be forward. \square

Multiversion Concurrency Control

- Historically, systems which work with just a single version of the database were widely used in DBMSs.
 - This model is called *single-version concurrency control (SVCC)*.
- Nowadays, however, a much more common approach is *multiversion concurrency control (MVCC)*.
- In MVCC, there may be several versions of each primitive data object.
- Exactly one version for each data object is committed, and belongs to the *global DB* – the “true” database which other transactions can see.
- The others are local copies for transactions.
- All updates by transactions (before commit) are to local copies.



Modelling Isolation under MVCC via Conflicts

- Common MVCC levels may be characterized by two parameters.

Effective time assignment: RRWE or RBWE.

Admissibility of concurrent edges: Only $f:rw$, $b:rw$, $f:ww$, $f:wr$ possible.

Policy	Eff. Time Assign.	Admissibility of concurrent edge type			
		$f:rw$	$b:rw$	$f:ww$	$f:wr$
RC	RRWE	Permitted	Permitted	Permitted	Permitted
SI	RBWE	Permitted	Permitted	Prohibited	Impossible
SIW	RBWE	Permitted	Permitted	Permitted	Impossible

- First assume that all transactions run at the same level of isolation.

Read Committed (RC): RRWE + all edge types allowed.
READ COMMITTED in PostgreSQL.

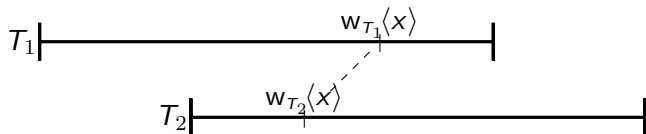
Snapshot Isolation (SI): RBWE + $f:ww$ prohibited; $f:wr$ impossible.
REPEATABLE READ in PostgreSQL.

Snapshot Isolation with concurrent writes (SIW): RBWE; $f:wr$ impossible.

Question: How can this be extended to a mixed-mode setting?

Winners and Losers — FCW and FUW

- In a prohibited conflict, only the *winner* may commit.



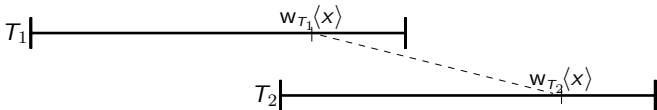
First committer wins (FCW):

First updater wins (FUW): Use request time; for ww-conflicts only.

- Widely used in practice, including PostgreSQL.
- In mixed mode, the *loser* transaction may not have a prohibited edge.

Policy	Eff. Time Assign.	Admissibility of concurrent edge type <i>for loser</i>			
		<i>f:rw</i>	<i>b:rw</i>	<i>f:ww</i>	<i>f:wr</i>
RC	RRWE	Permitted	Permitted	Permitted	Permitted
SI	RBWE	Permitted	Permitted	Prohibited	Impossible
SIW	RBWE	Permitted	Permitted	Permitted	Impossible

Examples of Mixed-Level Isolation



- Under both FCW and FUW, T_1 is the winner, T_2 the loser.
- (Isolation(T_1) = RC), (Isolation(T_2) = SI) \Rightarrow T_2 not allowed to commit.
- (Isolation(T_1) = SI), (Isolation(T_2) = RC) \Rightarrow both may commit.
 - The loser transaction, running under RC, plays by its own set of rules, which do not prohibit such concurrent writes.

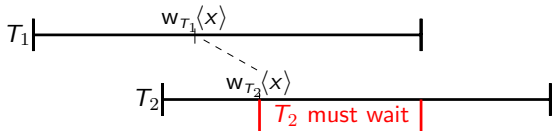
Observation: It is *not* always the case that running a transaction under SI will prevent concurrent writes of a data object.

Real world: This is how PostgreSQL (and other systems) implement mixed-level isolation.

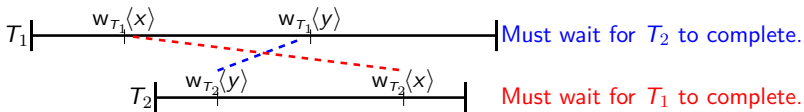
- Note that the write by T_2 is not even known when T_1 commits.
- Any “fix” would require that the transaction manager override the local isolation policy of the loser.

ShareLocks in PostgreSQL

- PostgreSQL implements write-write isolation via the *ShareLock*.
 - Essentially an exclusive (write, X-) lock.
- Whenever a transaction makes a write request, it must hold or be issued a ShareLock before it may continue.
 - If it cannot obtain such a lock, it must wait.
 - All such locks are held until the transaction commits or aborts.
 - This applies even to transactions running under RC.



This can also result in deadlock (which the system resolves).



Relationship to SQL-Defined Isolation

Anomaly table: Recall the *anomaly table* for the SQL standard.

Isolation Level	Admissibility of anomaly			
	Dirty Write	Dirty Read	Fuzzy Read	Phantom
READ UNCOMMITTED	Prohibited	Permitted	Permitted	Permitted
READ COMMITTED	Prohibited	Prohibited	Permitted	Permitted
REPEATABLE READ	Prohibited	Prohibited	Prohibited	Permitted
SERIALIZABLE	Prohibited	Prohibited	Prohibited	Prohibited

READ UNCOMMITTED: This isolation level *permits* reading of uncommitted data, but does **not** require it (my interpretation).

- Reading uncommitted data is not supported by most MVCC systems.
- READ UNCOMMITTED is often implemented as READ COMMITTED on MVCC systems.

READ COMMITTED: This definition makes sense in MVCC as well as SVCC.

- RC under MVCC, as defined in this work, is an acceptable match.


Relationship to SQL-defined isolation — 2

REPEATABLE READ: Examined in detail in [Berenson et al. 1995 SIGMOD].

- Correct semantics depends upon fine details of formalism.


Loose interpretation (for SVCC): $t_{\text{Read}\langle x \rangle} \langle T_1 \rangle < t_{\text{Write}\langle x \rangle} \langle T_2 \rangle < t_{\text{End}} \langle T_1 \rangle$
forbidden.

Equivalent (for MVCC): $t_{\text{Read}\langle x \rangle}^{\tau_1} \langle T_1 \rangle < t_{\text{Write}\langle x \rangle}^{\tau_2} \langle T_2 \rangle < t_{\text{End}} \langle T_1 \rangle$ forbidden
($\langle T_1, \tau_1 \rangle$ and $\langle T_2, \tau_2 \rangle$).

 But then readers can block writers and conversely.

- This is why many (including [Berenson et al. 1995 SIGMOD]) argue that SI is not a compliant implementation of REPEATABLE READ.

Opinion: REPEATABLE READ is poorly defined for use with MVCC, with SI is the most appropriate equivalent in that case.

 The REPEATABLE READ of the standard also allows phantoms, which is a bad idea according to [Berenson et al. 1995 SIGMOD].

- The REPEATABLE READ definition should exclude phantoms.
- The local definition of SERIALIZABLE is essentially REPEATABLE READ without phantoms. (More later)

Isolation Levels of SQL

Global scope: Recall that serializability is a *global* property, of a *set* of transactions.


- It does not make sense to say that a single transaction is serializable.

Question: How does one integrate serializable, as an isolation level, with local levels such as RC and SI?

Double-duty strategy: The (apparent) intent of the SQL standard was to give SERIALIZABLE double duty.

Local duty: Provide so-called *DEGREE 3* isolation (REPEATABLE READ+ no phantoms).

Global duty: If all transactions are run under SERIALIZABLE, the result should be serializable behavior.

 Unfortunately, DEGREE 3 isolation implies serializability only under very narrow constraints, requiring both SVCC and 2PL (more later). 😞

Goal: Realize this double-duty strategy in another way, which is compatible with modern MVCC.

Definitions of Serializability

- There are two main concepts of serializability in use.

View serializability: The best theoretical definition.

Drawback: Testing is NP-complete.

Conflict serializability: Defined by the absence of cycles in the DSG.

Advantage: Simple algorithm for testing.

Advantage: Easy to extract an equivalent serial execution.

- Just expand the partial order defined by the DSG to a total order (on the transactions).

Drawback: Slightly less general than view serializability; excludes some serializable behavior.

- The two agree in the absence of blind writes.

Convention: Conflict serializability is widely used in transaction modelling.

- It will be used in this work.

Serializable-Generating and Serializable-Preserving Strategies

Serializable Generating (SerGen): An isolation level is **SerGen** if, whenever all transactions are run at that level, the result is a serializable schedule.

 SerGen does not apply in a mixed-level setting.

Serializable Preserving (SerPres): An isolation level is **SerPres** if committing a transaction at that level does not add any new cycles to the DSG, *regardless of the level at which the previously committed transactions were run.*

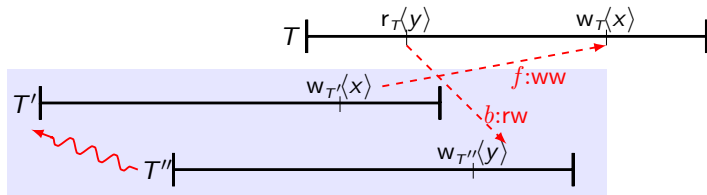
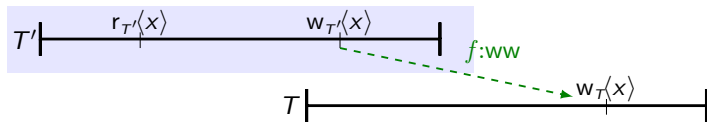
Observation: SerPres \Rightarrow SerGen. \square

SERIALIZABLE Policy	DBMS	SerGen	SerPres
SSI	PostgreSQL	Yes	No
SS2PL	SQL Server	Yes	?
SI	Oracle, MySQL/MariaDB	No	No
None	IBM Db2	N/A	N/A

Question: Are there useful SerPres strategies?

RCX and SIX: Examples of SerPres Isolation Levels

Observation: An isolation level which prohibits backward edges is SerPres. \square



RCX, SIX, SIWX: Examples of SerPres Isolation Levels

New SerPres local isolation levels:

Policy	Eff. Time Assign.	Admissibility of concurrent edge type <i>for loser</i>			
		<i>f:rw</i>	<i>b:rw</i>	<i>f:ww</i>	<i>f:wr</i>
RCX	RRWE	Permitted	Prohibited	Permitted	Permitted
SIX	RBWE	Permitted	Prohibited	Prohibited	Impossible
SIWX	RBWE	Permitted	Prohibited	Permitted	Impossible

Note: Prohibiting *b:rw* is all that is needed to achieve SerPres.

Advantage of RCX, SIX, SIWX: They solve the *mixed-scope* problem.

- They provide a well-defined local isolation level, which may be mixed with other levels in an understandable way (providing SerPres).
- When all transactions are run under any mix of RCX, SIX, or SIWX, true conflict-serializable isolation (SerGen) is obtained.

RCX and SIX in Practice

SERIALIZABLE Policy	DBMS	SerGen	SerPres
SSI	PostgreSQL	Yes	No
SI	Oracle, MySQL/MariaDB	No	No
SS2PL	SQL Server	Yes	?
SIX	Pyrrho, StrongDBMS	Yes	Yes
RCX	?	Yes	Yes
SIWX	?	Yes	Yes

Drawback: SIX involves strictly more false positives than SSI.

- RCX and SIWX are incomparable to SSI in this regard.

Advantage: RCX, SIX, SIWX provide meaningful isolation semantics in a mixed-level setting, *with simple semantics and implementation*.

Question: Are RCX, SIX, SIWX “good enough” in practice?

- The answer must come from benchmarking.
- Pyrrho seems to perform quite well.

Bottom line: RCX and SIX deserve further investigation as alternatives for implementing SQL SERIALIZABLE isolation.

Implications for Systems Employing SIX for Serialization

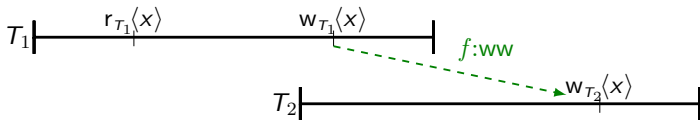
ww-conflicts: It is not necessary to exclude ww-conflicts to ensure serializability.

- SIWX will serve just as well as SIX.

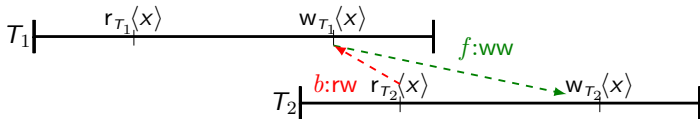
Question: Does it matter in practice?

Answer: Only to the extent that blind writes are present.

- A conflict excluded by SIX but not SIWX:

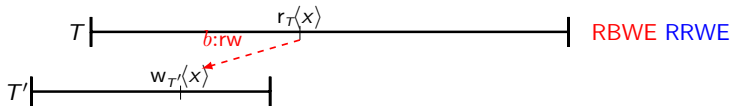
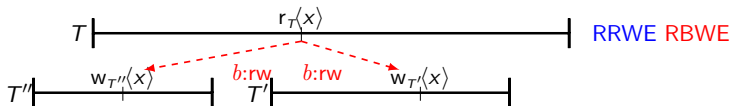


- A conflict excluded by both SIX and SIWX.



Superiority of RCX

Theorem: RCX (as a serialization strategy) involves (strictly) fewer false positives than does SIWX or SIX. \square



Implications for Systems Employing SIX/SIWX for Serialization

RBWE vs. RRWE: SIX or SIWX may be replaced with RCX with strictly fewer false positives and serializability retained.

In other words: do the following.

- Ignore ww-conflicts.
- Read at (first) request time rather than at the beginning of the transaction.
- If T reads x and a concurrent transaction T' writes x , tag that as an inadmissible edge iff
 - (a) $t_{\text{Read}(x)}^{\text{Req}}\langle T \rangle < t_{\text{End}}\langle T' \rangle$ (not $t_{\text{Read}(x)}^{\text{RBWE}}\langle T \rangle = t_{\text{Start}}\langle T \rangle < t_{\text{End}}\langle T' \rangle$).
 - (b) $t_{\text{End}}\langle T' \rangle < t_{\text{End}}\langle T \rangle$.

Optimality for Commit-Order Preservation

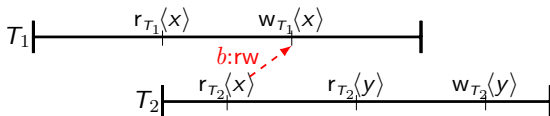
Definition: A serialization strategy AugTest is *commit-order preserving* if for any schedule S which AugTest approves as serializable, the order in which the transactions in S commit is an equivalent serial order.

Example: A serialization which is not commit-order preserving.

Constraint: $x + y \geq 500$

T_1 : $x \leftarrow x + 50$;

T_2 : If $x + y \geq 600$ then $y \leftarrow y - 100$;



Equivalent serial order: $\langle T_2, T_1 \rangle$.

Theorem: A serialization strategy is commit-order preserving iff it disallows all backward edges. \square

Corollary: RCX, SIX, SIWX are all commit-order preserving in any mix. \square

Corollary: RCX is optimal for commit-order preservation. \square

SSI — Serializable SI

- SSI is used in PostgreSQL to implement true conflict-serializable isolation.

Essential dangerous structure (EDS): $\langle T_2, \tau_2 \rangle \xrightarrow{-:-} \langle T_1, \tau_1 \rangle \xrightarrow{b:rw} \langle T_0, \tau_0 \rangle$.

- T_0 commits first.
- $T_1 \parallel T_2$. ($T_0 \parallel T_1$ automatic since edge is backward.)
- $\langle T_0, \tau_0 \rangle$ and $\langle T_2, \tau_2 \rangle$ may be the same.

Theorem: If the DSG contains a cycle, then it also contains an EDS.

Proof sketch:



Strategy: Check for and disallow EDSs.

Remark: If the base isolation level is SI, then $\langle T_2, \tau_2 \rangle \xrightarrow{-:rw} \langle T_1, \tau_1 \rangle$.

- But the base strategy does not depend upon SI.
- RC or SIW may be used just as well.

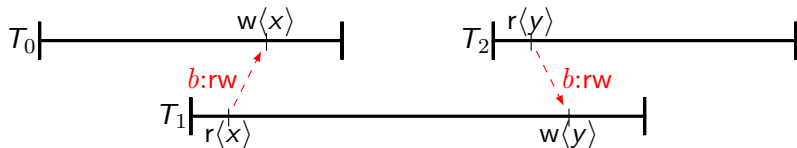
EDS and Locality

Local property of a transaction A property of a transaction T is *local* if it depends only upon T and those other transactions which run concurrently with T .

Examples: RC, SI, SIW, RCX, SIWX, and SIX are all defined in terms of local properties.

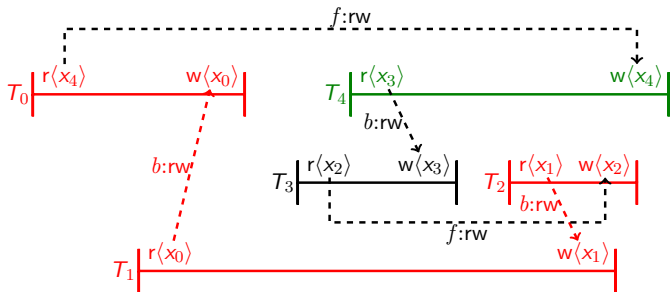
Fact: Participation in an EDS is not a local property. \square

Counterexample: In the diagram below, $\langle T_2, \tau_2 \rangle \xrightarrow{-:-} \langle T_1, \tau_1 \rangle \xrightarrow{b:rw} \langle T_0, \tau_0 \rangle$ forms an EDS for $\{\tau_0, \tau_1, \tau_2\} \in \{\text{RRWE}, \text{RBWE}\}$.



- But T_0 commits before T_2 begins.

SSI is not Serializable Preserving



- $\langle T_2, \tau_2 \rangle \xrightarrow{b:rw} \langle T_1, \tau_1 \rangle \xrightarrow{b:rw} \langle T_0, \tau_0 \rangle$ is the only EDS.
- Commit order is $\langle \langle T_0, \tau_0 \rangle, \langle T_3, \tau_3 \rangle, \langle T_1, \tau_1 \rangle, \langle T_2, \tau_2 \rangle, \langle T_4, \tau_4 \rangle \rangle$.
- Only when $\langle T_4, \tau_4 \rangle$ commits does a cycle appear.

⚠ So, if $\{\langle T_i, \tau_i \rangle \mid 0 \leq i \leq 3\}$ are run under REPEATABLE READ, and $\langle T_4, \tau_4 \rangle$ under SERIALIZABLE, a new cycle will be added upon the commit of $\langle T_4, \tau_4 \rangle$.

👎 Commit of $\langle T_4, \tau_4 \rangle$ is not blocked by SSI in this case.

SSN — Serializable Safety Net

SSN: Recently [Wang et al., VLDB J. 2017], a new approach to serialization called *Serializable Safety Net (SSN)* has been developed.

- It is similar to SSI in idea, but employs a more complex notion of “dangerous structure” ..
.. for which there nevertheless exist efficient algorithms for identification.
- It may be applied to essentially any system whose isolation levels incorporate write-at-end behavior.

Not SerPres: While it is serializable generating, it is unfortunately not serializable preserving.

Observations Regarding Serializability and the SQL Standard

Anomaly table: Recall the *anomaly table* for the SQL standard.

Isolation Level	Admissibility of anomaly			
	Dirty Write	Dirty Read	Fuzzy Read	Phantom
READ UNCOMMITTED	Prohibited	Permitted	Permitted	Permitted
READ COMMITTED	Prohibited	Prohibited	Permitted	Permitted
REPEATABLE READ	Prohibited	Prohibited	Prohibited	Permitted
SERIALIZABLE	Prohibited	Prohibited	Prohibited	Prohibited

Misconception: The standard does **not** regard the table as a definition of SERIALIZABLE.

SerGen: It is stated explicitly that SERIALIZABLE must be SerGen.

- So Oracle and MySQL/MariaDB do not satisfy the standard.

Sufficiency of the absence of anomalies: The absence of anomalies is a sufficient condition for serializability only under very strict conditions.

- SVCC with lock-based implementation of isolation.
- REPEATABLE READ uses “loose interpretation” (essentially SS2PL).
- Local isolation SERIALIZABLE = REPEATABLE READ + no phantoms (DEGREE 3).

Remarks Concerning the Terminology ACID

ACID: In the literature, assertions that a DBMS is *ACID* are often made.

Atomicity: All-or-nothing effect of transactions.

Consistency: Transactions are correct when run individually.

Isolation: As discussed in this talk.

Durability: Committed data are to permanent storage.



Isolation is a matter of degree, not an absolute all-or-nothing concept.

Question: How much isolation is “enough”?

- Conflict serializable?
- Commit-order-preserving serializable?
- The term ACID must be qualified to have unambiguous meaning.

Conclusions and Further Directions

Conclusions: Two models have been developed for transaction isolation.

Mixed-level model: for local-scope isolation (RC, SI).

- Provides a firm foundation for understanding what to expect when different transactions are run at different levels of isolation.

Mixed-scope model: for serializable isolation.

- Extends the global semantics of a serializable schedule by providing meaningful semantics (serializable preserving) to individual transactions running with isolation `SERIALIZABLE`,
- Even when others are running at other levels of isolation.

Further Directions:

- Experimental studies of the efficacy of RCX, SIWX, and SIX.
- Extension of the theoretical model to classical lock-based levels of isolation (e.g., SS2PL).