Transaction Isolation in Mixed-Level and Mixed-Scope Settings

Stephen J. Hegner
DBMS Research of New Hampshire, USA
dbmsnh@gmx.com

ADBIS 2019
Bled, Slovenia
10 September 2019

Isolation of Database Transactions

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

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

Isolation: Concurrent transactions should not interfere with each other.

Serializable schedules: The "gold standard" for isolation is *serializability*.

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

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

Reality: DBMSs offer a variety of isolation levels.

Tradeoff:
- Higher isolation ⇒ reduced concurrency.
- Lower isolation ⇒ 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.

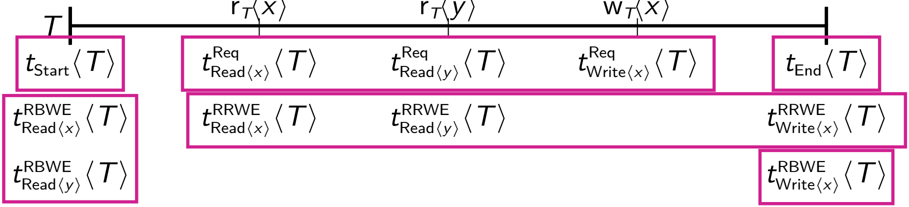
- They are properties of an individual transaction, depending only upon that transaction and its relationship to concurrent neighbors.

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.
  - $\tau = \text{RRWE} =$ *read (at) request write (at) end*.
  - $\tau = \text{RBWE} =$ *read (at) beginning write (at) end* (*snapshot read*).

Conflict Classification

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

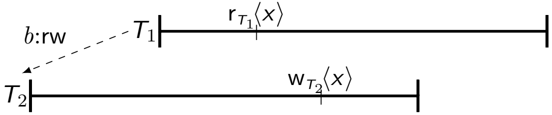
$\langle T_1, \tau_1 \rangle \xrightarrow{\text{rw}} \langle T_2, \tau_2 \rangle$: T_1 reads some x and T_2 is the next writer of x.

$\langle T_1, \tau_1 \rangle \xrightarrow{\text{ww}} \langle T_2, \tau_2 \rangle$: T_1 writes some x and T_2 is the next writer of x.

$\langle T_1, \tau_1 \rangle \xrightarrow{\text{wr}} \langle T_2, \tau_2 \rangle$: T_1 is the last writer of some x before T_2 reads x.

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

Backward edge: $\langle T_1, \tau_1 \rangle \xrightarrow{b:\text{zz}} \langle T_2, \tau_2 \rangle$ 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. □

Modelling Isolation via Conflicts

- Common MVCC levels may be charaterized by two parameters.

  Effective time assignment: RRWE (read at request, write at end)
  RBWE (read at beginning, write at end)

  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

- First consider the single-mode situation, in which all transactions run at the same level of isolation.

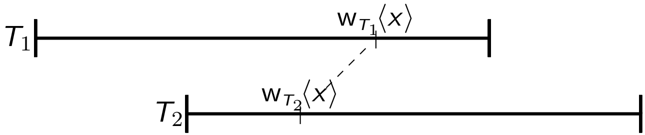
Read Committed (RC): RRWE + all edge types allowed.
`READ COMMITTED` in PostgreSQL.

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

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):

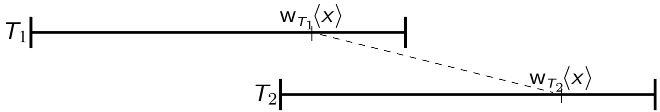
First updater wins (FUW): Use request time; for ww-conflicts only.

- 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 *for loser*			
		f:rw	b:rw	f:ww	f:wr
RC	RRWE	Permitted	Permitted	Permitted	Permitted
SI	RBWE	Permitted	Permitted	Prohibited	Impossible

Examples of Mixed-Level Isolation



- Under both FCW and FUW, T_1 is the winner, T_2 the loser.
- (Isolation$\langle T_1 \rangle$ = RC), (Isolation$\langle T_2 \rangle$ = SI) $\Rightarrow$ T_2 not allowed to commit.
- (Isolation$\langle T_1 \rangle$ = SI), (Isolation$\langle T_2 \rangle$ = 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 that 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.

Serializability Issues

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.

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

☟ Unfortunately, running all transactions with `DEGREE 3` isolation does not ensure serializable behavior. ☹

Goal: Realize this double-duty strategy in another way.

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 create any new nonserializable behavior, *regardless of the level at which the previously commited transactions were run*.

Conflict serializability: No cycles in the *direct serialization graph* (*DSG*), defined by rw-, ww-, and *wr*-conflicts.

Observation: SerPres ⇒ SerGen. □

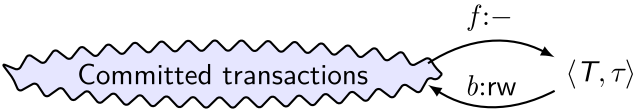
SERIALIZABLE Policy	DBMS	SerGen	SerPres
SSI	PostgreSQL	Yes	No
SI	Oracle, MySQL/MariaDB	No	No

Question: Are there useful SerPres strategies?

RCX and SIX: Examples of SerPres Isolation Levels

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

DSG:



New SerPres local isolation levels:

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

Note: In RCX, prohibiting b:rw is <u>all</u> that is needed to achieve SerPres.

Advantage of RCX and SIX: 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 RCX or SIX, they provide true conflict-serializable isolation (SerGen).

RCX **and** SIX **in Practice**

Use in practice: The RDBMSs Pyrrho and StrongDBMS employ SIX to implement `SERIALIZABLE` isolation.

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

Drawback: SIX involves strictly more false positives than SSI.

- RCX is incomparable to SSI in this regard.

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

Question: Are RCX and SIX "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.

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 and SIX.
- Extension of the theoretical model to classical lock-based levels of isolation (*e.g.*, SS2PL).