an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

Generalized Isolation Level Definitions

FEBRUARY 25, 2016

tags: Consistency, Transaction processing

Generalized Isolation Level Definitions (http://bnrg.cs.berkeley.edu/~adj/cs262/papers/icde00.pdf) – Adya et al. 2000

Following on from yesterday's **critique of ANSI SQL isolation levels (https://blog.acolyer.org/2016/02/24/a-critique-of-ansi-sql-isolation-levels/)**, today's paper also gives a precise definition of isolation levels – but does so in a way that is inclusive of optimistic and general multi-version concurrency control strategies instead of being dependent on locking. Where Berenson et al. define phenomena P0..P3 in order to explain the ANSI SQL isolation levels (degree 1, 2, and 3), Adya et al. define *generalised* phenomena labelled G0, G1, G2 whose definitions are independent of implementation strategy. Based on these generalised phenomena they then provide *portable isolation level* definitions (PL) that accommodate all implementation strategies, not just locking based ones.

> *We now show that the preventative approach is overly restrictive since it rules out optimistic and multi-version implementations. As mentioned, this approach disallows all histories that would not occur in a locking scheme and prevents conflicting operations from executing concurrently… The real problem with the preventative approach is that the phenomena are expressed in terms of single-object histories. However, the properties of interest are often multi-object constraints. To avoid problems with such constraints, the phenomena need to restrict what can be done with individual objects more than is necessary. Our approach avoids this difficulty by using specifications that capture constraints on multiple objects directly. Furthermore, the definitions in the preventative approach are not applicable to multi-version systems since they are described in terms of objects rather than in terms of versions. On the other hand, our specifications deal with multi-version and single-version histories.*

Following a motivation section, the authors first build up a model of a database system, and then define different types of conflict (read and write dependencies, as well as something called an anti-dependency that we'll get too) that can occur within that model. This leads to the central notion of a Direct Serialization Graph (DSG) for a history. The phenomena and isolation levels are then specified as constraints on the allowable graphs.

How to construct a Direct Serialization Graph

A DSG has one node for every *committed* transaction. (Directed) edges between these nodes represent read/write/anti-dependency conflicts. A transaction T1 *depends* on T2 if there is a path in the graph from T1 to T2. It *directly depends* on T2 if there is a edge from T1 to T2. In the description that follows, a delete is modeled as committing a special "dead" version of an object.

To construct a DSG start with one node for every committed transaction, and for all pairs of distinct nodes T1 and T2, add a read dependency edge from **T2** to **T1** if any of the following conditions hold:

> T1 commits a version x_i of object x, and T2 subsequently reads x_i.
>
> T1 commits a change, and T2 then performs a predicate-based read such that the set of objects matched by the predicate are altered by T1's commit. Furthermore, T1 is the *most recent* transaction to have committed a change impacting T1's matches.

Next add an anti-dependency edge from T2 to T1 if any of the following conditions hold:

> T1 reads some version x_i of object x, and T2 then commits the next version of x in the version history.
>
> T1 performs a predicate based read. T2 then commits a later (the next?) version of some object that would change the matches of T1.

And finally add a write dependency edge from T2 to T1 if

> T1 commits version x_i of some object, and T2 then commits the next version of x in the version history.

Phenomena and Isolation Levels

Level	Phenomena disallowed	Informal Description (T_i can commit only if:)
PL-1	G0	T_i's writes are completely isolated from the writes of other transactions
PL-2	G1	T_i has only read the updates of transactions that have committed by the time T_i commits (along with PL-1 guarantees)
PL-2.99	G1, G2-item	T_i is completely isolated from other transactions with respect to data items and has PL-2 guarantees for predicate-based reads
PL-3	G1, G2	T_i is completely isolated from other transactions, i.e., all operations of T_i are before or after all operations of any other transaction

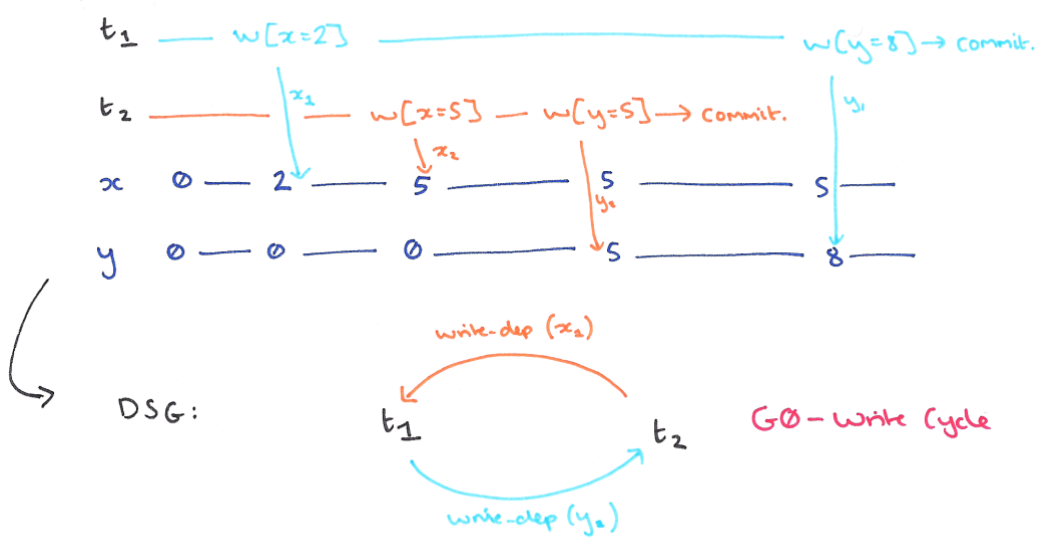
Figure 6. Summary of portable ANSI isolation levels

(https://adriancolyer.files.wordpress.com/2016/02/portable-isolation-levels-fig-6.png)

Isolation Level PL-1 (Portable definition of Read Uncommitted)

Phenomena P0 prevents dirty writes, and is motivated by a requirement to be able to serialize transactions based on writes, and to simplify recovery from aborts. The latter reason is not applicable to all systems, that may use other schemes for recovery. Portable Isolation level PL-1 provides for serialized transactions based on writes by disallowing *Write Cycles*.

A Write Cycle is phenomenon **G0**. A history exhibits a write cycle if its direct serialization graph contains a cycle consisting entirely of write-dependency edges. Note that this definition neatly encompasses write cycles with more than 2 objects involved.



(https://adriancolyer.files.wordpress.com/2016/02/write-cycle.png)

Isolation Level PL-2 (Portable definition of Read Committed)

The essence of no dirty reads – without being overly restrictive on the reading and writing of objects written by transactions that are still uncommitted – is captured by phenomenon G1, which is comprised of three sub-phenomena G1a, G1b, and G1c. G1 subsumes G0, and PL-2 is the isolation level that prohibits G1 (and hence G0 also).

> **G1a** – *Aborted Reads*. T2 reads some object (including via predicates reads) modified by T1, and T1 aborts. To prevent aborted reads, if T2 reads from T1 and T1 aborts, T2 must also abort – known as a *cascading abort*.
> **G1b** – *Intermediate Reads*. T2 reads a version of some object (including via predicate reads) modified by T1, and it was not T1's final modification of that object. To prevent intermediate reads, transactions can only be allowed to commit if they have read the final versions of objects from other transactions.
> **G1c** – *Circular Information Flow*. The Direct Serialization Graph contains a directed cycle consisting entirely of (read and write) dependency edges. If T1 is affected by T2, then there is no path by which T2 can also affect T1.

> *Proscribing G1 is clearly weaker than proscribing P1 since G1 allows reads from uncommitted transactions…. Our PL-2 definition treats predicate-based reads like normal reads and provides no extra guarantees for them; we believe this approach is the most useful and flexible.*

Isolation Level PL-2.99 (Portable definition of Repeatable Read)

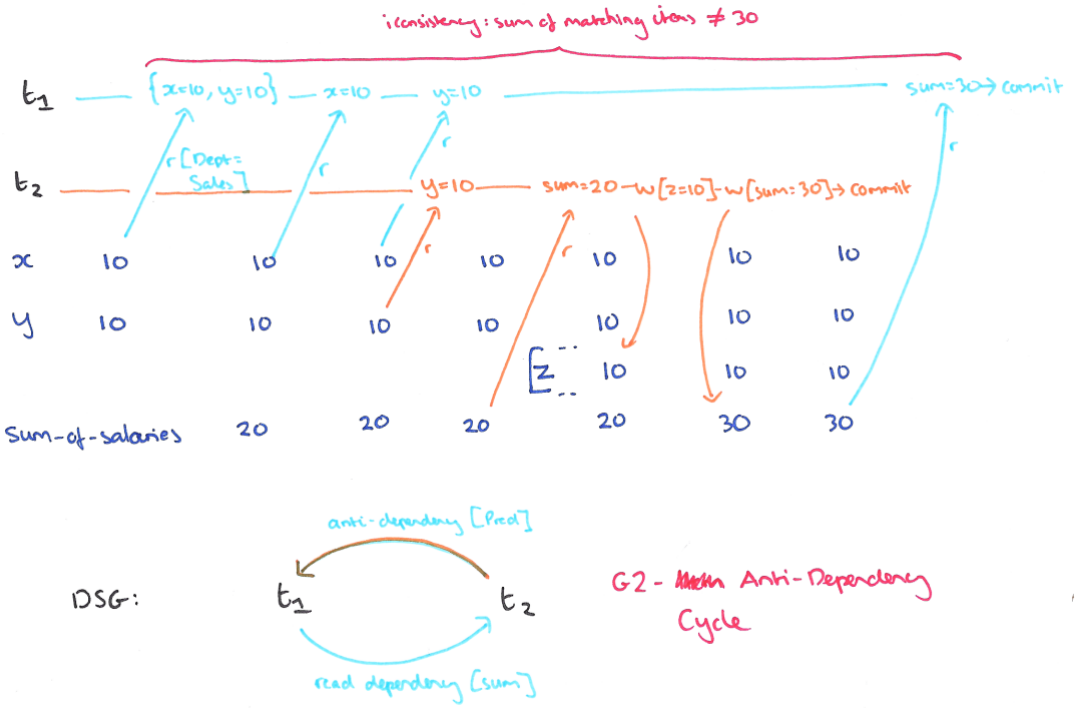
> *The level called Repeatable Read… provides less than full serializability with respect to predicates. Anti-dependency cycles due to predicates can occur at this level.*

Phenomenon **G2-item**, *Item Anti-Dependency Cycles* occurs when a DSG contains a directed cycle having one or more *item* anti-dependency edges.

PL-2.99 forbids G2-item anomalies in addition to G1 (and hence G0).

Isolation Level PL-3 (Portable definition of Serializable)

Consider the following history and DSG:



(https://adriancolyer.files.wordpress.com/2016/02/anti-dependency-cycle.png)

> *When T1 performs its query, there are exactly two employees, x and y, both in Sales. T1 sums up the salaries of these employees and compares it with the sum-of-salaries maintained for this department. However, before it performs the final check, T2 inserts a new employee, z2, in the Sales department, update the sum-of-salaries, and commits. Thus when T1 reads the new sum-of-salaries value it finds an inconsistency.*

This history is allowed at isolation level PL-2.99, but forbidden at isolation level PL-3, which prohibits phenomenon G2:

G2 *Anti-dependency cycles* occurs when a DSG contains a directed cycle with one or more anti-dependency edges (item or predicate).

> *Proscribing G2 is weaker than providing P2, since we allow a transaction to modify object x even after another uncommitted transaction has read x…. our specification for PL-3 provides conflict-serializability (this can be shown using theorems presented in [9]). All realistic implementations provide conflict-serializability; thus our PL-3 conditions provide what is normally considered as serializability.*

Mixing Isolation Levels

> *In a mixed system, each transaction specifies its level when it starts and this information is maintained as part of the history and used to construct a mixed serialization graph or MSG. Like a DSG, the MSG contains nodes corresponding to committed transactions and edges corresponding to dependencies, but only dependencies relevant to a transaction's level or obligatory dependencies show up as edges in the graph… For example, an anti-dependency edge from a PL-3 transaction to a PL-1 transaction is an obligatory edge since overwriting of reads matters at level PL-3.*

If an MSG is acyclic, and phenomena G1a and G1b do not occur for PL-2 and PL-3 transactions then a history is *mixing correct*. Such a history provides each transaction with the guarantees that pertain to its level.

Broader Applicability

> *Our approach is applicable to other levels in addition to the ones discussed in the paper. We have developed implementation-independent specifications of commercial isolation levels such as Snapshot Isolation and Cursor Stability, and we have defined a new level called PL-2+; the details can be found in [1 – Adya's PhD thesis]. PL-2+ is the the weakest level that guarantees consistent reads and causal consistency; it is useful in client-server systems and broadcast environments.*

from → Uncategorized

2 Comments leave one →

Trackbacks

1. ACIDRain: concurrency-related attacks on database backed web applications | the morning paper
2. Seeing is believing: a client-centric specification of database isolation | the morning paper