# Technical Perspective: Allocating Isolation Levels to Transactions in a Multiversion Setting

Alan D. Fekete
University of Sydney
alan.fekete@sydney.edu.au

Among the ways a database management system adds value, is the transaction abstraction, where the application coder can group together multiple data accesses that collectively perform one meaningful real-world activity. The platform will provide the "ACID" properties (atomic, consistent, isolated and durable) so the whole transaction happens like a single event. The mechanisms that allow this appearance include crash recovery and rollback (usually based on log entries) and concurrency control (typically involving locks).

To capture the essential goal of concurrency control, we consider whether a given execution of the system is "serializable", which means that there is another execution which is not-interleaved at all, and has the same outcome for both final state of the data, and the values observed in each transaction. In a serializable execution, any integrity property is guaranteed to hold at the end, even if it is not explicitly enforced by the platform, as long as each program running alone would preserve that integrity. The ideal is that every execution of the platform will be serializable.

An early mechanism for concurrency control was 2-Phase Locking, based on an update-in-place approach to data, and taking exclusive locks (for data modification) or shareable locks (for reads) and holding them till the end of the transaction. Many recent platforms instead use *multiversion* mechanisms where some accesses work on an older version of a data item, rather than the latest version; this can allow better performance because a read might proceed even before a concurrent writer has completed.

From early days of database technology, platforms have offered in the API a command that the application can invoke to SET TRANSACTION ISOLATION LEVEL to SERIALIZABLE, REPEATABLE READ, or READ COMMITTED. Gray et al [1] introduced Read Committed isolation as a variation on locking where sharable locks are held only during a read access, rather than till the end of the transaction as in traditional serializable 2-Phase Locking. Doing this may improve performance, but the data integrity could be at risk. There are also variations on multi-version concurrency control which give less isolation than serializability, though the system behaviors allowed at a given level are not exactly the same as what happens with the traditional locking-based ways.



The expectation of weak isolation, is that the application coder will decide which isolation level each transaction should use, based on their understanding of what is safe to do. Making this decision in a reasoned way, for the multiversion isolation mechanisms, is what is tackled by the recent work of Vandevoort, Ketsman and Neven. The way this work defines whether an allocation is safe to use for particular transactions (called *robust*), is to ask that every possible execution of those transactions when each is run with the chosen isolation level, will be serializable. Determining whether a given allocation is robust naturally leads to the question of finding the best allocation which is robust.

The work of Vandervoort et al is done in the style of theoretical computer science; there is a focus on precise definitions, and knowing whether a particular calculation takes time which is polynomial in the size of the input, or worse. There are as always, some simplifications in the theory, which keep the mathematics and notation manageable. This work extends a long tradition of theory applied to concurrency control [2]. Among the notable contributions of the paper by Vandervoort et al, is to capture what executions are possible for a given allocation. This is important because the isolation levels are typically described as if all transactions have the given isolation. Understanding how an execution can proceed with mixed isolation, requires careful looking at the implementation, and then abstracting out the interactions to state the key properties. The paper also offers a very nice theorem, that shows that if an allocation allows an interleaving of the transactions which is non-serializable, then there exists an interleaving of a particularly simple form, which the authors call *split schedule*. This allows one to test for robustness by checking far fewer interleavings, and is the heart of how this paper gets polynomial-time algorithms.

## 1. REFERENCES

[1] J. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger. Granularity of locks and degrees of consistency in a shared data base. In *Modelling in Data Base Management Systems, Proceeding of the IFIP Working Conference on Modelling in Data Base Management Systems, Freudenstadt, Germany, January 5-8, 1976*, pages 365–394, 1976. available at http://jimgray.azurewebsites.net/JimGrayPublications.htm.

[2] G. Weikum and G. Vossen. *Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*. Morgan Kaufmann, 2002.