Technical Perspective: When is it safe to run a transactional workload under Read Committed?

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

A data management platform provides many capabilities to assist the data owner, application coder, or end-user. For example, it should support an expressive query language, schema definition, and sophisticated access control. Another way many platforms add value is through a transaction mechanism, which allows the application programmer to indicate that a stretch of code, including multiple accesses to data, represents a single real-world activity and so all these steps should happen as if a single step, despite really being interleaved with other programs, or perhaps cancelled after partial execution. If the platform perfectly hides interleaving of different activities, the execution is called *serializable*, and this is a great aid to protecting data quality. Any integrity constraint over the data (whether explicitly declared in schema or not) which is preserved by each transaction running alone, is also valid at the end of any serializable execution of several transactions.

A traditional mechanism to ensure serializable executions is for the platform to isolate transactions, preventing one from interfering with another's accesses, by taking (and enforcing) transaction-duration locks on behalf of the user code. Some platforms maintain multiple versions of each item, and use these to get better performance for reading, because they can sometimes allow read access (to an older version) even when the item is locked by a writer transaction. But every mechanism we have for guaranteed serializability, leads to substantial reduction in throughput when there is too much contention by writers. From early days of relational DBMSs, platforms have allowed the application coder to chose a lower level of isolation [1], for example by releasing some locks early; the execution may then not appear perfectly like a transaction being a single step, but better performance is possible. "Read Committed" isolation level is indeed the default in most commercial and open-source platforms, and therefore many applications use it, without the programmer necessarily understanding the implications for correct (or otherwise) behaviour of their code.

In general, a lower isolation level can allow non-serializable execution and violation of data integrity; however, sometimes, one can have an application and an isolation mechanism, such that properties of the code or data imply that all executions will be serialiable, even though other applications at this isolation level can be non-serializable. One says that the application is *robust* for the given isolation level. The article by Vandevoort *et al.*, presents important results of this kind, for the widely-used Read Committed mechanism. These could have substantial practical impacts. When an application is shown to be robust, the programmer can get the improved performance of lower isolation, without risking data corruption. Knowing these results can also inspire programmers to design their code in ways that are robust. Maybe, in future tools might be built that automatically apply the theory to check robustness of application code.

Proving results about systems requires a way to model concepts from the world (such as the protocol, the code, etc) with mathematical structures such as sets and sequences. The theory of transaction management [2] has used a variety of models. Each model has simplifications (where things that are not identical in the world are treated the same) and restrictive assumptions (which simply refuse to model some situations that can really occur). For example, the model used in many of Jim Gray's early papers, treats the execution as a sequence of read and write operations, each coming from a transaction. This does not deal with the possibility of a *predicate read* such as SELECT-WHERE retrieving those items that meet some condition of their contents (rather than accessing on an unchanging primary identifier). Vandevoort *et al.* also have restrictions, but their work goes beyond the previous approaches in at least one important way: they include in their model, the possibility to have some constraints between items which can be accessed in the application (for example, constraints can arise from some foreign key properties in the schema). The paper elegantly shows how different features of the constraint set, can lead to different computational difficulty in deciding whether or not a set of programs is robust for Read Committed isolation.

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.