Problems in Providing Declarative Integrity Constraints and Semantic Optimization in Database Systems

Adrian Hudnott
adrianh@dcs.warwick.ac.uk

Integrity constraints are an essential component of database design, yet declarative support for constraints in contemporary systems is severely limited, which forces users to write tricky and inflexible procedural code to maintain database consistency. Modern business applications hold large amounts of data about each individual or organization, and with more complex schemas come more complex integrity constraints. There is a need for an efficient method of enforcing a wide variety of declarative constraints that can be applied to realistic databases without significantly degrading their performance. *The Third Manifesto* by Hugh Darwen and C.J. Date prescribes such support for declarative integrity constraints in a relational database system, but its prescriptions raise several research problems, each of which is linked to integrity constraints and beginning with enforcing the constraints. In the presentation I will describe these problems in further detail and show how they interrelate to each other and merit a common solution. The theme is the dependencies between data and the area is automated theorem proving for relational database applications.

Sometimes an update command can violate a constraint if it is interpreted in isolation, even though the overall transaction is valid. Contemporary SQL systems support "deferred constraint checking" to handle these situations. However, deferred constraint checking imposes the additional difficulty that the author of a user-defined operator cannot assume that the database is consistent upon entry to that operator. The *Manifesto* solves this by introducing simultaneous assignment instead of deferred checking. Simultaneous assignment is common in formal methods, but the usual implementation by copying into a temporary variable is too slow for large relations. I am developing an algorithm based on syntactic analysis and multiversion timestamps to address this problem.

Once integrity constraints have been checked the system must decide what to do with any invalid data. Usually transactions that leave the database inconsistent will be rolled back. However some compensating actions to repair the transaction may be desired, thereby making it an acceptable one after all. An example of a declaration requesting compensating actions is `ON DELETE CASCADE` in SQL. Finding workable sequences of compensating actions is called the transaction repair problem.

A database can contain "views", which are expressed as a formula over other data in the database. We may choose to cache the value of a view, which increases performance at the expense of introducing a cache coherence problem. I show that the task of keeping such "materialized views" up to date is a generalization of the constraint checking problem. Views can also be updated, which requires using the inverse of a view's defining formula to translate the requested update into an equivalent update to the base data. I show that this problem is an extension of the transaction repair problem.

The main activity of a DBMS beyond storing and updating data is processing users' queries. Database query languages are very expressive and constraint checking may involve executing one or more queries. Therefore methods that speed up querying are also beneficial to efficient constraint checking. I consider optimizations of queries that take advantage of constraint conformance as the kind of optimization of interest, because of the symbiotic relationship to enforcing the constraints and because this new kind of query optimization further demonstrates the usefulness of declarative constraints.

The most famous problem concerning dependencies between data in relational databases is deriving facts when some fields are unknown. SQL provides NULL for this purpose. However three-valued logic does not match human understanding of what it means for data to be unknown. I hope that by modeling missing data as free variables the forthcoming results for constraint checking will be able to derive genuine properties of missing data items that can be subjected to querying.

Finally, the *Manifesto* has a second usage of constraints– for defining subtypes. In object-oriented languages a constructor call always returns an object of the class where the constructor is declared. However under the *Manifesto* type system the counterpart of a constructor, known as a "selector", can return a value of any subtype of the type requested. This presents difficulties in efficiently identifying the runtime type of a variable in order to choose between multiple operator implementations. My research includes a study of the *Manifesto* type system with the intention of addressing this problem.

References

[1] Darwen, H., and Date, C. J. *Databases, Types, and the Relational Model: The Third Manifesto*, 3rd ed. Pearson Education, USA, 2006.