Integrity Control in Advanced Database Systems

Paul W.P.J. Grefen  Rolf A. de By  Peter M.G. Apers
Design Methodology & Database Groups
Computer Science Department, University of Twente
{grefen,deby,apers}@cs.utwente.nl

1 Introduction

Integrity control is a key feature of systems that provide access to large and shared data resources. Especially when such resources are updatable, integrity constraints have, as their prime use, the role of defining the system’s criteria for update validity. For complete integrity control we identify the following tasks:

- **Integrity specification and integration** by which we mean stating declaratively what constitutes a legal, i.e. semantically allowed, database state and also what should be done if constraints are invalidated.

- **Transaction verification** which aims at proving that a database transaction cannot possibly invalidate a given constraint.

- **Integrity enforcement** which deals with the way the system checks whether constraints have been invalidated by a transaction and “repairs” integrity violations.

The first two tasks can be considered “specification-time” of an application, whereas the last task is a run-time notion of the system. Fully functional integrity management should entail all three activities, but most current systems – be they CASE-tools or DBMS’s – provide little to support them.

In this paper, we give an overview of our research on integrity control in advanced database systems. We have been following two major tracks in this respect. The first is embedded in the PRISMA project on parallel, main-memory, relational databases [1]. Integrity enforcement has been an active area of research in this project, and we discuss its results in Section 2. The second track has been dealing with integrity integration and transaction verification in the context of a functional object-oriented database model, called TM[2]. This work is described in Section 3. In the Conclusions we will argue that our two tracks have been independent but orthogonal approaches to the integrity management problem that we hope to combine in the future.

2 Integrity control in a parallel DBMS

In the PRISMA project, a parallel main-memory relational database system with complete functionality has been developed [1]. One of the research tracks in the project has investigated the design and integration of an integrity control subsystem that fits the requirements of a high-performance database system. In this section, we discuss the approach taken and the results obtained in this research track.

When integrating integrity control mechanisms into a full-blown database system, a number of aspects are important. The integrity control algorithms should have clear semantics at the right level of abstraction. In particular, they should take transaction semantics into account. Also, the integrity control mechanisms should provide complete functionality, i.e., they should deal with a broad range of integrity constraint types. Further, the mechanisms should be implementable in a modular fashion with well-defined interfaces, such that integration into the DBMS architecture is relatively easy. And last but not least, the integrity control subsystem should offer good
performance, as the cost of integrity control is generally considered one of the main obstacles for its general use in practice.

2.1 Approach

To obtain both clear semantics and complete expressiveness, constraints should be specified in a high-level declarative language like first-order logic. The constraint that every employee should be employed in an existing department can thus be specified as follows, for example:

\[(\forall x)(x \in \text{Employee} \Rightarrow (\exists y)(y \in \text{Department} \land x.(\text{department} = y.(\text{name})))\]

Constraints must be translated to a high-level language with operational semantics, fit for use in constraint enforcement by the DBMS. In the PRISMA approach, an extension of the relational algebra has been chosen as constraint enforcement language. This language, called XRA, has a formal definition and clear semantics [13]. The above constraint can be expressed as follows in XRA\(^1\):

\[\text{alarm}(\pi_{\text{department}} \text{Employee} - \pi_{\text{name}} \text{Department})\]

Constraints translated to their extended relational algebra form can be used to modify a transaction such that the resulting transaction is safe, i.e. cannot violate any constraints. This approach to integrity control, called transaction modification, is used in the PRISMA database system. Its clear semantics are supported by a declarative specification of the algorithms [11, 12], which can be easily mapped to a modular system architecture [9]. The modification of a transaction is performed by analyzing the transaction on a syntactical basis and extending it with XRA statements for integrity control purposes. This implies that the actual effect of the transaction on the database does not have to be taken into account. This gives the approach a performance and complexity advantage over other approaches, but also implies a risk of indefinite triggering behaviour in the integrity control subsystem [11, 12].

Modified transactions can be executed in the same parallel fashion as “unmodified” transactions. In the PRISMA system, this means that several forms of parallelism are available [8, 11]. Multiple transactions can be executed in parallel, each having their own processes for transaction and integrity control, and relational operation execution. Multiple layers of the DBMS can work in parallel on the same transaction, passing XRA statements through a “preprocessing pipeline”. This means for example that transaction modification and transaction execution can proceed in parallel. Further, multiple XRA statements in one transaction and multiple XRA operations in one statement can be executed in parallel. And finally, as PRISMA uses fragmented relations, intra-operator parallelism can be used to spread the work of one relational operator over multiple processors.

2.2 Results

A prototype transaction modification subsystem has been integrated into the PRISMA database system, resulting in a full-fledged system with integrity control mechanisms. Experiments with this system have been performed on a POOMA shared-nothing multi-processor hardware platform. Performance measurements on this machine have resulted in two important observations that we discuss below.

Parallelism has shown to be an effective way to deal with the high processing requirements of integrity control on large databases, and transaction modification has shown to be an adequate technique to support parallelism. Detailed performance measurements are presented in [10, 11]; here we only present a typical example to give the reader an idea of the obtained results. Given is a benchmark database with a key relation of 1000 tuples and a foreign key relation of 10000 tuples, both of the well-known Wisconsin type. Enforcing a referential integrity constraint after insertion of 1000 tuples into the foreign key relation takes less than 2 seconds on an 8-node POOMA system. Enforcing a domain constraint in the same situation only takes a few tenths of a second.

\(^1\)The \text{alarm} operator aborts the transaction in which it is embedded if its operand is non-empty.
The second observation is that the costs of integrity constraint enforcement in terms of execution times are quite reasonable compared to the costs of executing “bare” transactions [10]. Depending on the type of the constraint and the data fragmentation, the overhead of constraint enforcement ranges from a few percent to about the execution time of the bare transaction. This implies that integrity control is indeed usable in real-world systems with large amounts of data.

3 Integrity control in an OODB

3.1 Data model context

The TM data model is a functional, object-oriented database specification language [2] based on Cardelli-Wegner type theory. For database purposes it has been extended with logic and set expressions [4], and is currently being revised to suit the definition of technical information systems in the ESPRIT-project IMPRESS (EP 6355). It serves as the vehicle for our study of OODB-related issues, which comprises specification, optimization, and implementation. We believe that our data model is a clean, non-procedural language in the style of, for instance, the $O_2$ data model.

The TM data model allows the specification of classes and a special database object. For each of these, one may declare typed attributes, first-order logic constraints and methods. Constraints as defined in classes come in two flavours: object and set. An object constraint is a restriction on the possible values of the attributes of each object of the class in isolation\(^2\). A set constraint is a rule over the set of all objects of the class. This distinction is not intrinsic and serves ease of use, or in other words, each object constraint can be described as a set constraint. Constraints can also be defined on the database object, and all constraints together define a set of allowed database states, the so-called database universe [3].

The object-set distinction is also made for methods defined in a class: an object method takes a single object of the class as argument, a set method takes a set of such objects. This set need not be the full population of the class but can be a subset of it. A database method, furthermore, is an operation on the database object. An orthogonal distinction between methods (of any kind) is that of retrieval and update. A retrieval method simply extracts information from its arguments, whereas an update method can be used to (destructively) change its first argument. In this sense, a database retrieval method is a query and a database update method is a transaction. Needless to say, methods may invoke other methods as long as some natural laws of encapsulation are respected.

The result of a TM specification is the definition of (1) a single database object, (2) its allowed values, and (3) the allowed operations on it, adhering to rules of encapsulation. This means that if we want to provide an operation on a single object in the database, we have to do so via the database object level and such provision can be done automatically.

3.2 Constraints and methods

The use of the TM data model assumes a simple ACID transaction model of integrity enforcement. In other words, inconsistent transactions can be specified and will be dealt with by run-time rollbacks. But in practice, this is an undesirable situation as, at least in principle, every constraint needs to be checked. This either results in poor performance or the complete abandoning of constraint enforcement. This problem has obviously been identified in database research a long time ago, and our approach to it is both methodological and through proof checking techniques. We discuss both issues.

\(^2\)Constraints on just a single object are also possible, but they would be phrased as object constraints that refer to some unique identification of the object.
3.3 Transaction correctness methodology

As the levels of granularity – i.e. object, set, database – of methods on the one hand and constraints on the other coincide, this is taken advantage of in the implementation of TM on real DBMS’s. This is work currently being carried out in the ESPRIT project IMPRESS. The TM language is implemented on top of an object-oriented programming language that has been enriched with persistency and an interface to an object server with complex object algebra. The programming environment and object server run in a client-server architecture.

In this implementation, for each object update method an adorned version is created that checks all object constraints of the class at hand. For each set update method an adorned version is created that checks both the object and set constraints of the class. Each database update method will take into account all constraints, addressing the issue of relevance obviously: some object, set and database constraints need not be verified due to the method code. For instance, a transaction that updates objects of a single class only, needs not check object and set constraints of other classes. Note that the method adornment technique has some similarities with the transaction modification approach described in Section 2. As a tiny example, consider the object constraint ‘self.age ≤ 65’ of the class Employee and an object method inc that increments the age of an employee object. The adorned version, basically saying not to change the object if the constraint would be violated afterwards, will look like:

```plaintext
object update method
  inc_adorned = if inc[self].age ≤ 65
                 then inc[self]
                 else self
    endif
```

An adorned method is created such that if it calls an update method, it calls its adorned version. In this way, constraints are treated in a structural and efficient way. As much as possible, constraint checks are translated to preconditions on the actual data manipulation to increase run-time efficiency.

Checking any constraint in a method results by default in correctness or abort, but this is an over-simplified approach. Our method allows designers to define corrective actions upon expected constraint invalidation. This can be done on a per-method and per-constraint basis and may result in several adorned method versions, depending on application requirements [7].

3.4 Transaction verification

To further reduce the number of constraints to be checked by a transaction one can try to prove that the transaction can never invalidate a constraint and thus that checking it is needless.

We are currently working on proof checking techniques for compile-time transaction verification in the style of [14]. Our work is a proper extension of their work as it sets out to deal with proof checking issues on a theory in which subtyping and parametric types form an integral part. A simple first version has been implemented in the Cambridge HOL system [5], and we are currently revising that system based on lessons learnt [6].

4 Conclusions

To make integrity control techniques usable for database practice, attention should be paid both to issues of functionality and semantics, and to issues of feasibility and performance. As described above, the research at the University of Twente addresses both aspects. The work in the IMPRESS project focuses on functionality and semantics in a database system with an advanced data model. The work in the PRISMA project focuses on feasibility and performance in a database system with an advanced architecture.

In integrity control, compile-time transaction verification and run-time constraint enforcement are two complementary techniques. Combination of both techniques will provide the best basis for complete functionality
and high performance. Run-time constraint enforcement can deal with high performance requirements by the use of parallelism, such that integrity control on large databases becomes easily feasible.

References

[1] P.M.G. Apers et al.; PRISMA/DB: A Parallel, Main-Memory Relational DBMS; IEEE Trans. on Knowledge and Data Engineering; Vol.4, No.6, 1992.


