Parallel Handling of Integrity Constraints

Paul W.P.J. Grefen Jan Flokstra
Peter M.G. Apers
University of Twente

Abstract

Integrity constraints form an important part of a data model. Therefore, a complete integrity constraint handling subsystem is considered an important part of any modern DBMS. In implementing an integrity constraint handling subsystem, there are two major problem areas: providing enough functionality and delivering good performance in constraint enforcement. In the PRISMA project, an integrity constraint handling subsystem for a relational DBMS is developed, that meets both requirements. Functionality is reached through a modular and extensible architecture of the subsystem. Performance is reached through extensive use of parallelism in various constraint enforcement algorithms.

1 Introduction

In databases a part of the real world is modeled. The real world model consists of structures, operations, and constraints [Tsich82]. The structures represent the real-world entities and relations between them. The operations allow manipulation of the structures to model changes in the real world. The constraints guarantee the validity of the real world model represented by the database, by constraining the allowed operations on the database. As such, the constraints describe (part of) the semantics of this model [Tsich82], or, in other words, the knowledge of application properties [Simon87].

Especially in the relational data model integrity constraints are necessary to describe the semantics of the applications, because this data model has very little implicit semantics [Tsich82]. The growing complexity of modern database applications further increases the need for powerful constraint handling mechanisms. Therefore, a complete constraint handling mechanism is considered an important part of any modern DBMS, and is thus included in the PRISMA DBMS, called PRISMA/DB.

There are two general problem areas in implementing a constraint handling mechanism in a DBMS. At the first place, performance of constraint enforcement is a great problem. At the second place, providing a complete functionality is a problem. Solutions for these two problems are the goal of the research towards constraint handling in the PRISMA project:

- high performance of the constraint enforcement mechanism is achieved through extensive use of parallelism in various algorithms;
- complete functionality is reached through a very modular design of the constraint handling subsystem, guaranteeing a high degree of flexibility and extensibility.

[0]The work reported in this document was conducted as part of the PRISMA project, a joint effort with Philips Research Laboratories Eindhoven, partially supported by the Dutch "Stimuleringsprojectteam Informaticaonderzoek (SPIN)"

There has been some research in the field of integrity constraint handling already for centralized and traditional distributed database systems, e.g. [Stone75, Zloof78, Simon85, Morg84, Simon87]. In the context of parallel database systems the topic is new. This paper combines the theory of integrity constraint handling with the theory of query execution in parallel database systems, and adds some new ideas to obtain a full-fledged integrity constraint handling mechanism for parallel database systems with fragmented relations.

The paper starts with a discussion of the ideas that form the basis for constraint handling in PRISMA/DB; these ideas lead to the approach taken for design and implementation of the constraint handling subsystem in PRISMA/DB. Next, the architecture of the constraint handling subsystem is described in the context of the full DBMS architecture. Then, the various algorithms for constraint enforcement are discussed. The combination of these algorithms provides a good combination of functionality, flexibility and efficiency. The paper ends with some conclusions.

2 PRISMA approach to constraint handling

In this section the approach to constraint handling as taken in the PRISMA project is discussed. We start with a short discussion of the PRISMA/DB context. Next, the requirements to constraint handling are identified. From these, the approach chosen in the project follows.

2.1 The PRISMA/DB context

In the PRISMA project a parallel, main memory database management system is developed, called PRISMA/DB [Kers87, Apers88]. The system is designed to run on a shared-nothing multi-processor hardware architecture [Bron87]. To support parallelism in query execution, PRISMA/DB uses horizontally fragmented relations. Figure 1 shows the simplified base architecture of PRISMA/DB. The following components can be identified:

Data Dictionary (DD) the DD forms the central storage of all system information, such as information on relations and their fragments; the DD is also responsible for the creation and deletion of new fragments (OFMs);

Concurrency Controller (CC) the CC is responsible for the serializability of concurrent transactions; a two-phase locking protocol is used with fragments as the locking granularity [Date83];

User Interface (UI) the UI is the interface that enables interactive communication with one of the user language parsers of the system (here only the SQL interface is shown);

SQL Parser (SQL) the SQL parser translates SQL queries into the internal relational language of PRISMA/DB; further, the SQL parser informs the DD about the creation and deletion of relations;

Query Optimizer (QO) the QO deals with resolving fragmentation transparency, view removal, translation of recursive expressions, and, of course, the traditional optimization of queries;

Transaction Manager (TM) the TM manages the execution of schedules produced by the QO; for this purpose, the TM builds execution infrastructures out of OFMs and tuple transport channels; the TM also provides transaction serializability through locking and transaction atomicity through a two phase commit protocol [Date83];

One Fragment Manager (OFM) the OFM manages a single base fragment or intermediate result in the database; all fragment data is kept in main memory; disk storage is used for logging and checkpointing only; the OFM also executes all relational operators.

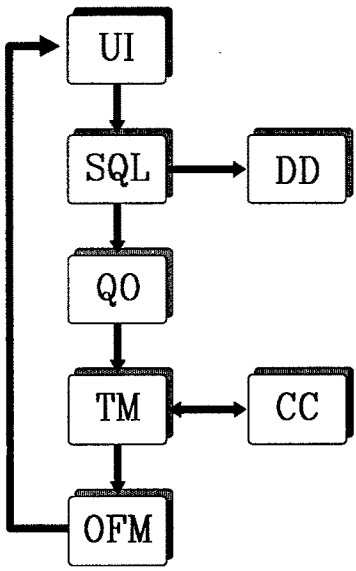

Figure 1: PRISMA/DB base architecture

An important role in PRISMA/DB is played by its internal relational language, called eXtended Relational Algebra (XRA) [Wils90a]. As its name suggests, XRA is an extension to the normal relational algebra; the extensions allow using the language as an operational language. XRA is used as the interface language for the interfaces between SQL Parser, Query Optimizer, Transaction Manager and One Fragment Manager.

2.2 Requirements

The PRISMA project provides an experimental research environment for the development of a DBMS. In this research two main issues can be identified. At the first place, the use of parallelism to improve the performance of a main memory relational DBMS is investigated. At the second place, the feasibility of the implementation of a full blown DBMS in a high level object oriented programming language is an issue under investigation.

The main goals in the field of integrity constraint handling can be deduced from these general project goals. At the first place achieving high performance through the use of parallelism in constraint enforcement is important. Further, using the implementation environment to obtain a complete functionality is considered a main topic. So, the constraint handling subsystem in PRISMA/DB has to meet the following requirements:

- the subsystem must be able to make extensive use of parallelism in the enforcement of constraints to obtain high performance;
- the subsystem should allow a high degree of expressiveness in constraints, to be able to model complex applications;
- as experimenting with the functionality is considered very important, a high degree of flexibility and extensibility should be available;
- as the base architecture of PRISMA/DB already exists, the subsystem must fit easily into the modular PRISMA/DB architecture;

- since integrity constraint handling is one of several research topics in the PRISMA context, the implementation effort for the subsystem should not be too large.

2.3 Approach taken

As lined out above, the approach to constraint handling as taken in the PRISMA project is based on performance on the one hand and on functionality and flexibility on the other hand.
To achieve high performance, a two track approach to parallelism is employed:

- parallelism in constraint enforcement can be obtained in the same way it is obtained in regular query execution if constraints can be translated to normal query constructs in XRA; in this case constraint enforcement is controlled explicitly by the Transaction Manager; therefore, we call this form of enforcement *explicit constraint enforcement*; this approach allows a high degree of functionality and flexibility;
- parallelism can be obtained by having a 'self-checking' data layer; in PRISMA/DB terms this means that the One Fragment Managers containing the base fragments enforce their constraints autonomously; this form of constraint enforcement is called *implicit constraint enforcement*; this approach allows a high degree of efficiency.

Good functionality and flexibility are achieved by having a strict decomposition of the tasks involved in constraint handling; the following tasks are identified:

- *translation* of constraints from the form as specified by the user to the form fit for dircet enforcement by the system;
- *storage* of constraints in both source as translated form;
- *enforcement* of the translated constraints within the transaction mechanism.

3 Architecture for constraint handling

The architecture of the constraint handling subsystem is deduced from the approach described in the previous section. Below we first describe how this architecture is integrated into the existing PRISMA/DB architecture. Next, attention is paid to the constraint translation module, the only fully new module needed for constraint handling.

3.1 Global architecture

As mentioned before, there are three important tasks to be performed for constraint handling in our approach:

- constraint translation
- constraint storage
- constraint enforcement

Because a two way approach to constraint enforcement is adopted in PRISMA/DB, we can split up the last task into explicit and implicit constraint enforcement. So, in total four tasks can be identified. These four tasks and their allocation in the PRISMA/DB architecture are discussed below. Figure 2 shows the modifications to the base architecture as shown in Figure 1.

Constraint translation is considered a fully new task with respect to the base architecture of PRISMA/DB. Therefore, a new component is designed for this task; this module is called the Constraint Compiler (abbreviated as C2, to avoid confusion with the abbreviation for the Concurrency

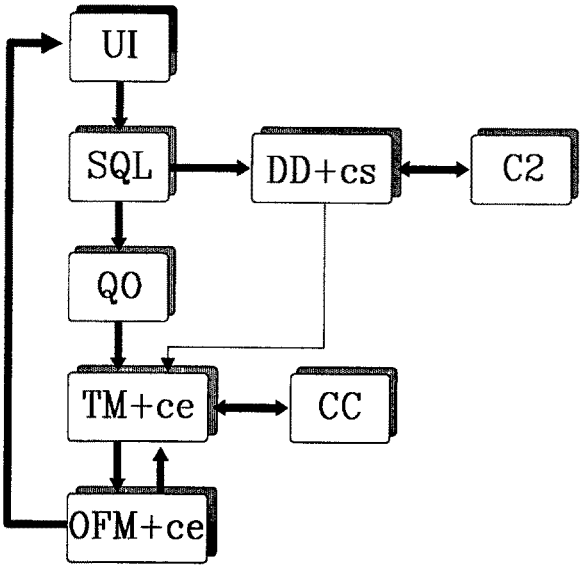

Figure 2: PRISMA/DB extended architecture

Controller component). The C2 component receives constraint specifications at the relation level in source format as its input from the Data Dictionary (DD), and returns optimized constraint specifications at the fragment level in internal format to the DD.

Constraint storage is considered part of the storage of data definitions. Since this task is already allocated with the Data Dictionary, constraint storage is also handled by this component. The DD receives constraint specifications at relation definition time and stores them. Further, the DD activates the C2 module to obtain constraint definitions in internal format. Performing the translation at constraint definition time avoids large overhead at constraint enforcement time. As shown in the figure, the DD is extended with a constraint storage module (cs).

Explicit constraint definitions are stated in XRA, just like ordinary queries. Therefore, it is obvious that explicit constraint enforcement is handled in the same way as regular queries. So explicit constraint enforcement is handled by the Transaction Manager (TM). The TM is extended with a constraint enforcement module (ce) for this purpose. The TM retrieves the constraint definitions in XRA format from the DD, as shown by the thin arc in the figure.

Implicit constraint enforcement is performed at the data level in the DBMS. Therefore, this task is allocated with the OFMs managing the base fragments in the system. The OFM component is extended with a constraint enforcement (ce) module. This module contains special purpose algorithms for local constraint enforcement. Constraint specifications are passed to the OFM at its creation time.

3.2 Constraint compiler

The Constraint Compiler (C2) performs the translation and optimization of constraints in external format at the relation level to constraints in internal format at the fragment level. Since PRISMA/DB employs two different kinds of constraint enforcement, there are two internal constraint formats:

- for explicit constraint enforcement, the constraints are translated to XRA constructs; these constructs are optimized to avoid query optimization overhead at constraint enforcement time; further, constraints are labeled with triggers that indicate when they have to be enforced to

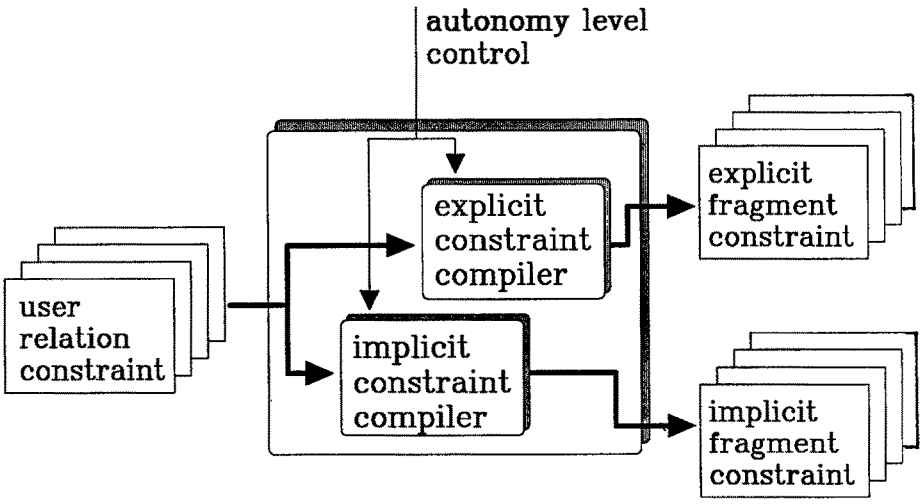

Figure 3: Constraint compiler architecture

avoid unnecessary constraint enforcement; the translation from external form to XRA is treated in detail in Sections 4.2 and 4.3;

- for implicit constraint enforcement, the constraints are translated to special purpose data structures for the OFM components; these data structures are interpreted by the OFMs; the translation from external form to OFM constraints is described in Section 5.1.

To obtain maximum flexibility, both translation types are fully separated in the C2 component. This may be inefficient, but since constraint translation is done statically (at constraint definition time only), performance is not an issue here. The way constraints are split up into explicit and implicit constraints is fully dynamically controllable by the user of the system. So, per relation one can decide what part of the constraints should be enforced explicitly or implicitly. The architecture of the constraint compiler is depicted in Figure 3.

4 Explicit constraint enforcement

For explicit constraint enforcement, constraints specified by the user are translated to XRA constructs; this involves the following two steps:

- translation of the constraints to the fragment level;
- mapping the translated constraints to XRA constructs;

The XRA constructs are executed by the Transaction Manager at the end of a transaction to enforce the constraints. The whole process of translating and enforcing explicit constraints is illustrated by two important classes of example constraints, domain constraints and referential integrity constraints; these constraints are presented below.

4.1 Example constraints

Constraints are denoted as a pair $I = [t, r]$, with the following elements:

- t is the set of *triggers* of the constraint; this set specifies the update types that may violate the constraint;

- r is the rule of the constraint; the rule is a boolean function with a part of the database schema as its domain.

A more formal description of this notation can be found in [Gref89, Gref90a].

Using this notation, a domain constraint is defined at the relation level as follows:

$$\begin{aligned} Il_{rel} &= [tl_{rel}, rl_{rel}] \\ tl_{rel} &= \{INS(R), UPD(R)\} \\ rl_{rel} &= (\forall x \in R.i)(c(x)) \end{aligned}$$

in which $c(x)$ is some boolean condition over x. The trigger set of this constraint states that constraint enforcement is necessary whenever a transaction performs an insert or update operation on relation R; clearly, a delete operation cannot violate a domain constraint. The rule of the constraint specifies that each value in attribute i of relation R has satisfy condiction c.

A referential integrity constraint as defined in [Date81] can be formulated in our notation as follows:

$$\begin{aligned} I2_{rel} &= [t2_{rel}, r2_{rel}] \\ t2_{rel} &= \{INS(R), UPD(R), DEL(S), UPD(S)\} \\ r2_{rel} &= (\forall x \in R.i \mid x \neq null)(\exists y \in S.j)(x = y) \end{aligned}$$

In this definition, $S.j$ is a key (unique attribute) of relation S. The constraint states that every foreign key in relation R that is not equal to null, must have its counterpart in relation S.

4.2 Translating constraints to the fragment level

Constraints defined by the user are formulated in terms of relations. Because enforcement of constraints takes place at the fragment level of the system, a translation to this level is necessary. The translation of constraints is comparable to the translation of queries from the relation to the fragment level [Ceri84]. The objective of the entire translation is to obtain a specification of the constraints that can straightforwardly be used for constructing efficient enforcement algorithms for the constraints.
The constraint translation is accomplished in the following steps:

- translation of the constraint at the relation level into canonical fragment form; this step brings the definition of the constraints from the external specification level (stated in terms of relations) to the internal level (stated in terms of fragments);
- distribution of the canonical form to the fragments; this step makes the semantics of the constraint fragment oriented; the result of this step is a set of constraints;
- optimization of the distributed fragment form; this step tries to obtain possibilities for a more efficient enforcement of the constraints;

These steps are treated in detail in [Gref89, Gref90a]. Here we limit ourselves to an illustration of the translation by means of the two example constraints.

4.2.1 Translation from relation to fragment level

Constraints specified in terms of relations have to translated to constraints specified in terms of fragments. The first step is translation to the canonical form. Assuming that relation R is fragmented into n fragments, the canonical form for domain constraint Il is the following:

$$\begin{aligned} Il_{can} &= [tl_{can}, rl_{can}] \\ tl_{can} &= \{INS(R_1), \ldots, INS(R_n), \\ &\quad UPD(R_1), \cdots, UPD(R_n)\} \\ rl_{can} &= (\forall x \in (R_1.i \cup \cdots \cup R_m.i))(c(x)) \end{aligned}$$

This constraint is defined in terms of fragments, but the semantics are still relation-oriented. To obtain fragment-oriented semantics, the canonical form is distributed to a set of fragment constraints, containing one constraint for each fragment in the relation. The constraint for fragment R_k obtained by distributing $I1_{can}$ is the following:

$$\begin{aligned} I1_{R_k} &= [t1_{R_k}, r1_{R_k}] \\ t1_{R_k} &= \{INS(R_k), UPD(R_k)\} \\ r1_{R_k} &= (\forall x \in R_k.i)(c(x)) \end{aligned}$$

The same steps can be applied to referential integrity constraint $I2$. The construction of the canonical fragment form is straightforward; the construction of the distributed fragment form is different, however. In constraint $I2$ two relations are involved that play different roles: introduction of new values in relation R may cause a constraint violation, whereas deletion of existing values in relation S can violate the constraint. Therefore, separate constraints are constructed for the fragments R_k of relation R and S_k of relation S:

$$\begin{aligned} I2_{R_k} &= [t2_{R_k}, r2_{R_k}] \\ t2_{R_k} &= \{INS(R_k), UPD(R_k)\} \\ r2_{R_k} &= (\forall x \in R_k.i \mid x \neq null) \\ &\quad (\exists y \in (S_1.j \cup \cdots \cup S_n.j))(x = y) \end{aligned}$$

$$\begin{aligned} I2_{S_k} &= [t2_{S_k}, r2_{S_k}] \\ t2_{S_k} &= \{DEL(S_k), UPD(S_k)\} \\ r2_{S_k} &= (\forall x \in (R_1.i \cup \cdots \cup R_m.i) \mid x \neq null) \\ &\quad (\exists y \in (S_1.j \cup \cdots \cup S_n.j))(x = y) \end{aligned}$$

The rule of $I2_{S_k}$ cannot be simplified in the general case, due to the fact that referenced values from S_k may be inserted again into another fragment of S within the same transaction (i.e. tuple migration between the fragments of a migration to keep the fragmentation consistent).

4.2.2 Optimization of constraints

The constraint definitions that are the results of the translation as described above are not very efficient for enforcement. Therefore, optimization of these constraints is necessary. Constraints at the fragment level can be optimized in a number of ways:

- the amount of data to be checked can be reduced by checking only those parts of fragments that have been changed in a relevant way; this method has already been described as differential test [Simon85, Simon87, Gard89];
- constraint rules can be algebraically manipulated to obtain forms that are cheaper in execution; this technique is similar to regular query optimization by expression rewriting [Ceri84];
- constraint rules can be simplified in some cases if knowledge about the fragmentation of relations is used; this is similar to removing 'empty' subtrees from query trees in query optimization [Ceri84].

As an example, we show how referential integrity constraint $I2_{R_k}$ as defined above can be optimized. The amount of data to be checked is reduced by replacing R_k by the differential set containing only the new values in R_k; this differential set is denoted as R_k^+. Next, the rule is manipulated by pushing the existential quantor through the union operation; this makes it possible to perform the existence check local to the fragments (opening the possibilities for parallelism). These optimizations result the following constraint definition:

$$\begin{aligned} I2_{R_k} &= [t2_{R_k}, r2_{R_k}] \\ t2_{R_k} &= \{INS(R_k), UPD(R_k)\} \\ r2_{R_k} &= (\forall x \in R_k^+.i \mid x \neq null) \\ &\quad (\bigvee_{w=1}^{w=n}((\exists y \in S_w.j)(x = y))) \end{aligned}$$

If relation R is fragmented using fragmentation constraints defined only on attribute $R.i$, and relation S is fragmented using the same fragmentation constraints defined only on $S.j$, we know that references from R_k are always to S_k; therefore, we can simplify the referential integrity constraint $I2_{R_k}$ as shown above to the following:

$$\begin{aligned} I2_{R_k} &= [t2_{R_k}, r2_{R_k}] \\ t2_{R_k} &= \{INS(R_k), UPD(R_k)\} \\ r2_{R_k} &= (\forall x \in R_k^+.i \mid x \neq null)(\exists y \in S_k.i)(x = y) \end{aligned}$$

It is clear, that enforcement of this constraint is much easier (and cheaper) than the enforcement of the constraint in its previous form. This leads to the observation, that integrity constraints should be taken into account in relation fragmentation design.

4.3 Translation to XRA constructs

Constraints are specified by the user in a non-procedural form. To be able to execute the constraints, they are translated to a procedural form in XRA. The XRA necessary for this is only a minimal extension to the XRA needed for normal query execution.

The use of XRA as an enforcement vehicle gives several important advantages over specialized hard coded algorithms:

- XRA provides an abstraction level that makes straightforward translation of constraints into enforcement algorithms possible;
- enforcement via XRA makes use of modular building blocks for the enforcement algorithms, thus ensuring flexibility and extensibility;
- for enforcement via XRA software building blocks are used that are already available for regular query processing to a large extent; this minimizes implementation overhead for integrity constraint handling on the one hand, and maximizes the use of parallel algorithms on the other hand.

4.3.1 Basic XRA constructs

For constraint enforcement in XRA, we make use of the regular relational algebra operators, such as union and difference. Further, we make use of a few extensions to the normal relational algebra. Apart from one, the *alarm* operator, all these extensions are also used for normal query processing in PRISMA/DB.

XRA contains two operators that take care of distributing tuples of a source operand to several destination operands; these operators are functionally equivalent to a combination of regular relational algebra operators, but, as is shown below, operationally extremely important for obtaining parallelism. The first of these operators, *copy*, copies the source operand to each of the destination operands:

$$copy\ (src, dst_1, \cdots, dst_n)$$

This operation is functionally equivalent to the following sequence of assignments:

$$\begin{aligned} dst_1 &\leftarrow src \\ &\vdots \\ dst_n &\leftarrow src \end{aligned}$$

The second distributing operator, *split*, splits up the source operand over the destination operands given the fragmentation constraints of the destination operands:

$$split\ (src, dst_1, cond_1, dst_2, cond_2, \cdots, dst_n, cond_n)$$

In this operation, all conditions $cond_i$ are mutually disjoint and together complete with respect to the source relation src; in other words, the conditions define a partition of the source relation. This operation is functionally equivalent to the following sequence of assignments:

$$\begin{aligned} dst_1 &\leftarrow \sigma_{cond_1} src \\ &\vdots \\ dst_n &\leftarrow \sigma_{cond_n} src \end{aligned}$$

Finaly, we have an operator that has as its sole functionality that it causes a transaction abort if its operand is not empty:

$$alarm\ (oper)$$

Using the XRA operators, logic constructs as appearing in constraint definitions can be translated to XRA constructs. Transformation rules as shown below are used for this. These rules give a few examples that can be used in the context of the example constraints of this paper.

$$(\forall x \in R)(c(x)) \rightarrow alarm\ (\sigma_{\neg c(x)}(R)) \quad (1)$$

$$(\forall x \in R)(\exists y \in S)(x = y) \rightarrow alarm\ (unique(R) - S) \quad (2)$$

$$(\forall x \in R)(\bigvee_{i=1}^{n}(\exists y \in S_i)(x = y)) \rightarrow \begin{array}{l} copy\ (unique(R), T_1, \cdots, T_n) \\ alarm\ ((T_1 - S_1) \cap \cdots \cap (T_n - S_n)) \end{array} \quad (3)$$

4.3.2 Translating constraints to XRA

Here we show how the example constraints can be translated to XRA constructs. Domain constraint $I1$ is taken as a first example; the definition of this constraint is:

$$\begin{aligned} I1_{R_k} &= [t1_{R_k}, r1_{R_k}] \\ t1_{R_k} &= \{INS(R_k), UPD(R_k)\} \\ r1_{R_k} &= (\forall x \in R_k^+.i)(c(x)) \end{aligned}$$

The rule of this constraint can easily be mapped onto the following XRA construct using transformation 1 as shown above:

$$alarm\ (\sigma_{\neg c(x)}(\pi_i(R_k^+)))$$

We take referential integrity constraint $I2_{R_k}$ as a second example; the form based on general fragmentation of the involved relations was described above as follows:

$$\begin{aligned} I2_{R_k} &= [t2_{R_k}, r2_{R_k}] \\ t2_{R_k} &= \{INS(R_k)\} \\ r2_{R_k} &= (\forall x \in R_k^+.i \mid x \neq null) \\ &\quad (\bigvee_{w=1}^{w=n}((\exists y \in S_w.j)(x = y))) \end{aligned}$$

Using transformations 1 and 3 as listed above, we can map the rule of this constraint onto the following XRA construct:

$$\begin{array}{l} copy\ (unique(\pi_i(\sigma_{i \neq null}(R_k^+))), temp_1, \cdots, temp_n) \\ alarm\ ((temp_1 - \pi_j(S_1)) \cap \cdots \cap (temp_n - \pi_j(S_n))) \end{array}$$

Note, that the *copy* operator takes care of distributing the differential set of R_k; this is used to obtain pipeling parallelism, as discussed below. The form of constraint $I2_{R_k}$ that was optimized with respect to matching fragmentation of the involved relations is the following:

$$\begin{aligned} I2_{R_k} &= [t2_{R_k}, r2_{R_k}] \\ t2_{R_k} &= \{INS(R_k), UPD(R_k)\} \\ r2_{R_k} &= (\forall x \in R_k^+.i \mid x \neq null)(\exists y \in S_k.j)(x = y) \end{aligned}$$

In this case, the resulting XRA construct can be:

$$split(unique(\pi_i(\sigma_{i \neq null}(R_k^+))), temp_1, f_1, \cdots, temp_n, f_n)$$
$$alarm(temp_1 - \pi_j(S_1))$$
$$\vdots$$
$$alarm(temp_n - \pi_j(S_n))$$

The differential set R_k^+ is not sent completely to every fragment of S, but is split up over the fragments of S, thereby reducing tuple transport; further, the costly intersection operation is not necessary here.

4.4 Enforcing constraints

The XRA constructs used for explicit constraint enforcement as presented in the previous section can straightforwardly be implemented using XRA execution infrastructures in PRISMA/DB. This section discusses the way this is realized and the possibilities for parallelism in this method.

4.4.1 Building the infrastructure

The infrastructure for constraint enforcement at the OFM level consists of three types of building blocks:

- permanent OFM : used for the storage of fragments of permanent relations (base fragments);
- temporary OFM : used for execution of relational operators and storage of intermediate results of operations;
- channels : used for the transportation of tuples between OFMs.

The permanent OFMs contain the fragment data on which integrity constraints must be enforced. These OFMs contain algorithms to automatically maintain the differential sets; this means that these sets are already constructed during transaction execution on a local basis in the fragments. The temporary OFMs are used for the execution of the XRA operators needed for the constraint enforcement structures. These OFMs can be created dynamically by the Transaction Manager when needed. The channels are used as communication means to transport tuples from one OFM to another. Both OFMs and channels are designed to make optimal use of pipelining in executing XRA [Wils89, Wils90b].

To obtain complete transaction semantics with respect to atomicity, all constraints are enforced at commit time in PRISMA/DB; note, that the techniques as presented in this paper can be used for other approaches equally well. Enforcing constraints at the end of a transaction consists conceptually of two phases:

- *setup phase*: in this phase the transaction manager builds the execution infrastructure needed for constraint enforcement; actually, this phase can already start during transaction execution;
- *execution phase*: in this phase the execution infrastructure processes the data to be checked; similar to the execution of regular user queries, this operates in a fully parallel, pipelined fashion [Wils90b].

The setup phase makes use of the same mechanisms that are used for setting up normal query execution infrastructures in PRISMA/DB; this implies that constraint enforcement does not require any architectural changes at the execution level [Gref90c]. Especially, the Transaction Manager does not have to deal with any part of the database extension, thus avoiding a possible bottleneck in the enforcement algorithms.

The previously presented example of the XRA construct used for enforcement of referential integrity constraint $I2$:

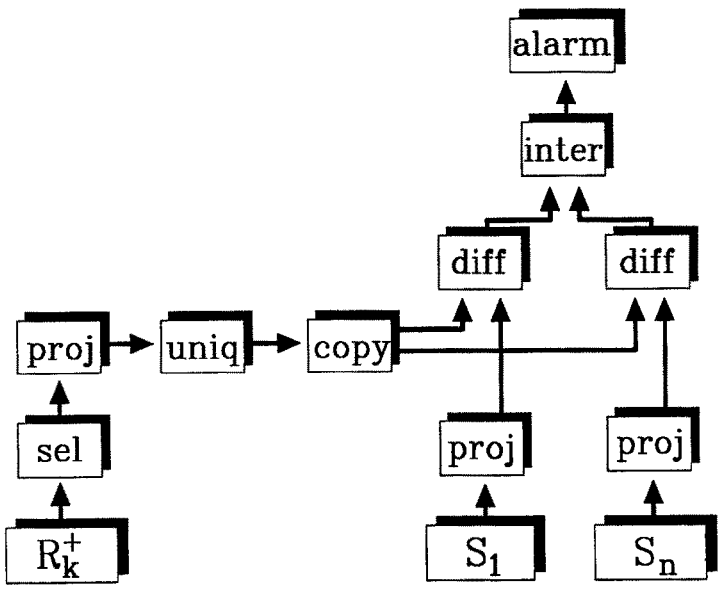

Figure 4: XRA constraint enforcement infrastructure

$copy\ (unique(\pi_i(\sigma_{i \neq null}(Rk^+))), temp_1, \cdots, temp_n)$
$alarm\ ((temp_1 - \pi_j(S_1)) \cap \cdots \cap (temp_n - \pi_j(S_n)))$

can straightforwardly be implemented by the execution infrastructure as shown in Figure 4.

4.4.2 Parallelism in explicit constraint enforcement

We have seen that constraint enforcement is executed by an XRA infrastructure. The relational operators in these infrastructures are all independent processes, so parallelism can be employed easily. Within the explicit constraint enforcement mechanism we can distinguish three types of parallelism [Gref88, Wils89]:

- several independent constraints can be checked at the same time; this is possible because after the setup phase, the enforcement process is fully asynchronous; at the constraint enforcement level, we can consider this a form of *multi-tasking*;
- at the same level in a XRA infrastructure, several OFMs operate on the data of several fragments in parallel; in the example of figure 4 we can see that all difference operators can work in parallel; this kind of parallelism is called *task spreading*;
- because all operators operate in a pipelined fashion, several stages of the XRA infrastructure can work in parallel too; we call this *pipelining* parallelism [Wils90b].

Further, improvements in response time for the entire transaction can be reached by executing normal queries of the transaction and constraint enforcement queries in parallel. As such, an overlap is created that shortens the overall transaction execution time. Take as an example a transaction that first insert some tuples into two distinct fragments managed by OFMs OFM_1 and OFM_2, and thereafter computes the join of both fragments on a third OFM $OFM3$. In this case the activity of the OFMs can be as depicted in Figure 5.

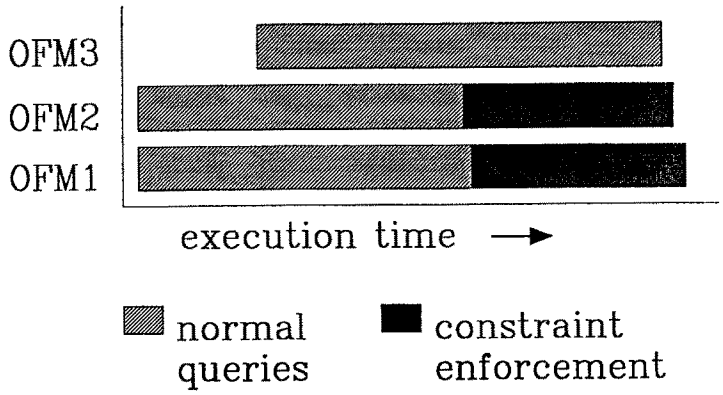

Figure 5: Regular query execution and explicit constraint enforcement

5 Implicit and hybrid constraint enforcement

In the previous section explicit constraint enforcement is discussed. This approach to constraint enforcement has a high degree of functionality and flexibility. This approach has however two drawbacks with respect to high efficiency:

- there is overhead at constraint enforcement time in creating the necessary XRA execution infrastructure and in passing the XRA commands to the One Fragment Managers involved in constraint enforcement;
- no specialized algorithms in the OFMs can be used for constraint enforcement to obtain high efficiency, because all constraints are stated in regular XRA.

To overcome these drawbacks, PRISMA/DB also employs implicit constraint enforcement. This form of enforcement is characterized by a high degree of autonomy of the involved One Fragment Managers: the OFMs enforce constraints without intervention of the Transaction Manager. Implicit constraint enforcement is however only applicable to very limited forms of integrity constraints. Therefore, PRISMA/DB employs a third technique which is a combination of explicit and implicit constraint enforcement; this form of enforcement is called hybrid constraint enforcement.

5.1 Translating constraints for OFMs

As shown in Figure 3 the compilation process in the Constraint Compiler component is directed by the autonomy level control. Four levels of OFM autonomy are distinguished by the C2 component [Gref90b]:

1. The lowest level of autonomy is called *no autonomy*; in this case all constraints are handled by *explicit enforcement* as described in the previous section; the OFM has no knowledge about constraints.
2. In the case of *strictly local* autonomy, the OFM enforces standard constraints having a strictly local scope with respect to the fragment managed by the OFM; this is a form of *implicit constraint enforcement.*
3. If *data reduction* autonomy is used, the OFM also offers special purpose differential sets for explicit constraint enforcement; the OFM constructs these sets by performing a filtering technique on the standard differential sets; this is a form of *hybrid constraint enforcement.*

4. The highest level of autonomy is called *process reduction* autonomy; in this case the OFM uses built-in logic, that can decide that explicit constraint enforcement for a specific constraint is not necessary, because the specal purpose differential set is empty; this is also a form of *hybrid constraint enforcement.*

Constraint sets for OFMs are produced by the Constraint Compiler in special purpose data structure format. These constraints are passed to the OFMs at their creation time by the Data Dictionary.

5.2 Implicit enforcement of constraints

In implicit constraint enforcement, an OFM enforces local constraints without any intervention from the TM. The constraint enforcement is triggered by the standard two-phase commit protocol; as such, the enforcement is treated as a part of the local commit decision making in the OFM and does not need any special communication with the TM. Implicit constraint enforcement can be applied to the following types of constraints:

- domain and nonull constraints: if the OFM is aware of the domains of the attributes of the fragment it manages, it can easily check if the values in the tuples of the fragment match with these domains;
- general tuple constraints: the same holds for constraints describing relations between values within one tuple;
- local uniqueness constraints: the OFM can enforce an uniqueness constraint if the relation to which the OFM belongs is not fragmented, or the relation is fragmented on the attributes on which the uniqueness constraint is defined.

5.3 Hybrid enforcement of constraints

In hybrid constraint enforcement, local activities of the OFM are used to alleviate the process of explicit constraint enforcement. The key to this is the maintenance of special purpose differential set for specific constraints. These differential sets can then be used in two ways:

- the sets can be smaller than the general purpose differential sets; this reduces the amount of data to be processed in explicit constraint enforcement; this used in *data reduction* autonomy;
- the OFM can check whether a special purpose differential set is empty, thereby making explicit constraint enforcement superfluous; this is used in *process reduction* autonomy.

The first of these protocols does not need any changes in the communication protocol between TM and OFM. The only necessary change at this level, is to make both TM and OFM aware of the fact that a special purpose differential set is used for a specific constraint. Because the Constraint Compiler generates both the information for TM and OFM, this is rather easy.

The process reduction protocol is somewhat more complicated. The TM has to request a status report from the OFM about the constraints used with process reduction. The OFM produces a status report indicating the constraints having non-empty differential sets. These constraints are then enforced by the TM in the usual way via XRA. And as usual, constraint violation will lead to a transaction abort. The necessary communication primitives between TM and OFM are depicted in Figure 6 [Gref90c].

We illustrate the idea of hybrid constraint enforcement with an example. Suppose we have a relation *Employer* of which the attribute *dept* is a foreign key referring to the attribute *name* of relation *Department*. Then we have the following constraint I:

$$
\begin{aligned}
I &= [t, r] \\
t &= \{INS(Employer), UPD(Employer), DEL(Department), UPD(Department)\} \\
r &= (\forall x \in Employer.dept \mid x \neq null)(\exists y \in Department.name)(x = y)
\end{aligned}
$$

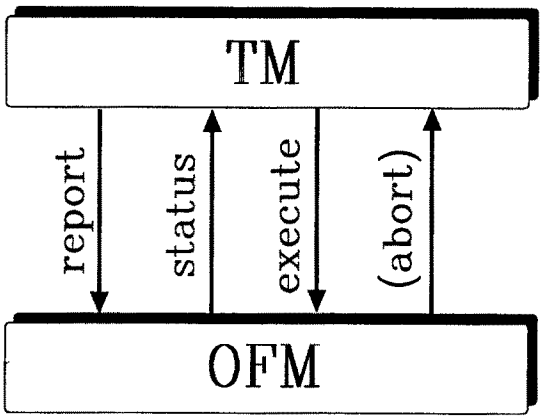

Figure 6: Communication for process reduction

To use hybrid enforcement for this constraint, all fragments of relation *Employer* are notified that attribute *dept* forms a foreign key in a referential integrity constraint. Now suppose fragment $Employer_k$ contains the following tuples:

name	*empnr*	*dept*
johnson	64576	sales
smith	64537	staff
richardson	24356	sales
crosby	48675	admin

If a transaction inserts two new tuples (jackson,34567,sales) and (dundee,76545,prod) into $Employer_k$, the general purpose differential set contains two tuples. If data reduction is used, the OFM can decide locally that the first of these two tuples can never violate the referential integrity constraint, because the foreign key value of this tuple is already present in the fragment. So, the amount of data to be checked in explicit constraint enforcement can be reduced. If a transaction inserts two new tuples (jackson,34567,sales) and (hilbilly,32224,admin), the general purpose differential set contains two tuples again. If data reduction is used, the OFM can decide locally that it can reduce the differential set for the referential integrity constraint to an empty set. If process reduction is also used, the OFM can inform the TM that the constraint does not need any explicit enforcement.

6 Conclusions

In this paper we have lined out an approach to parallel handling of integrity constraints on fragmented relations in a relational database system. The approach is designed such, that it meets both the requirements of a high degree of functionality and flexibility as well as high performance.

The first requirement is met by using a strictly modular design of the integrity constraint handling subsystem, in which a separate constraint compiler plays a central role, and by using regular query execution techniques for the enforcement of constraints. This approach results in a clear separation of constraint definition preprocessing, constraint enforcement protocol handling and constraint enforcement data processing.

The requirement of performance is met by having a two way approach to constraint handling that heavily uses the possibilities of parallelism. In *explicit constraint enforcement*, constraints are enforced using relational algebra operators, employing the same means for parallelism as in normal

query execution. In *implicit constraint enforcement*, a high degree of autonomy of the data management layer of the DBMS is used to obtain parallelism and to use special purpose algorithms. For maximum flexibility, a integration of both techniques is found in *hybrid constraint enforcement.*

The techniques have been implemented in the PRISMA/DB parallel main memory DBMS. The integration of the constraint handling subsystem has been easy due to the modular design of this subsystem. Currently, the subsystem supports the following types of constraints:

- domain constraints
- general tuple constraints
- nonull constraints
- uniqueness constraints
- referential integrity constraints

These constraint types cover the structural constraints of the relational data model [Gard89].

There are two main issues to be investigated in the future. In the first place, the performance of the various constraint enforcement protocols has to be evaluated. This will enable tuning the overall integrity constraint enforcement process. Further, the evaluation should make clear the advantages of parallelism in constraint enforcement in a quantitative sense. In the second place, the usability of the presented techniques for a broader range of constraint types has to be investigated, thus showing the general applicability of the approach.

Acknowledgements

We wish to thank the PRISMA project members for providing a challenging environment and productive cooperation with the teams from Philips Research Laboratories Eindhoven, the University of Amsterdam and the Centre for Mathematics and Computer Science Amsterdam in the development of our DBMS. In particular, Carel van den Berg is acknowledged for his ideas about local constraint enforcement in the One Fragment Manager component. Further, we wish to thank dr. A.J.Nijman for bringing academia and industry together, dr. H.H.Eggenhuisen for providing good project management and for stimulating the interaction between the various subprojects.

References

[Apers88] P.M.G. Apers, M.L. Kersten, H.C.M. Oerlemans; *PRISMA Database Machine: A Distributed Main Memory Approach*; Proceedings International Conference on Extending Database Technology; Venice, Italy, 1988.

[Bron87] W.J.H.J. Bronnenberg, L. Nijman, E.A.M. Odijk, R.A.H. v. Twist; DOOM: A Decentralized Object-Oriented Machine; IEEE Micro; October 1987.

[Ceri84] S. Ceri, G. Pelagatti; *Distributed Databases, Principles and Systems*; McGraw-Hill, 1984.

[Date81] C.J. Date; *Referential Integrity*; Proceedings of the 7th Conference on Very Large Data Bases; Cannes, France, 1981.

[Date83] C.J. Date; *An Introduction to Database Systems, Volume II*; Addison-Wesley, 1983.

[Gard89] G. Gardarin, P. Valduriez; *Relational Databases and Knowledge Bases*; Addison-Wesley, 1989.

[Gref88] A.N. Wilschut, P.W.P.J. Grefen, P.M.G. Apers, M.L. Kersten; *Implementing PRISMA/DB in an OOPL*; Memorandum INF 88-69; University of Twente, The Netherlands, 1988.

[Gref89] P.W.P.J. Grefen; *Integrity Constraint Handling in a Parallel Database System*; Memorandum INF 89-59; University of Twente, The Netherlands, 1989.

[Gref90a] P.W.P.J. Grefen, P.M.G. Apers; *Parallel Handling of Integrity Constraints on Fragmented Relations*; Proceedings DPDS'90; Dublin, Ireland, 1990.

[Gref90b] P.W.P.J. Grefen; *Design Considerations for Integrity Constraint Handling in PRISMA/DB1*; PRISMA Document P508; University of Twente, The Netherlands, 1990.

[Gref90c] P.W.P.J. Grefen, C. v.d. Berg; *PRISMA/DB1 TM-OFM Interface*; PRISMA Document P517; University of Twente, Centre for Mathematics and Computer Science, The Netherlands, 1990.

[Kers87] M.L. Kersten et al.; *A Distributed Main Memory Database Machine*; Proceedings of the 5th International Workshop on Database Machines; Karuizawa, Japan, 1987.

[Morg84] M. Morgenstern; *Constraint Equations: Declarative Expression of Constraints with Automatic Enforcement*; Proceedings of the 10th Conference on Very Large Data Bases; Singapore, 1984.

[Simon85] E. Simon, P. Valduriez; *Integrity Control in Ditributed Database Systems*; MCC Technical Report Number DB-103-85; MCC, Austin, USA, 1985.

[Simon87] E. Simon, P. Valduriez; *Design and Analysis of a Relational Integrity Subsystem*; MCC Technical Report Number DB-015-87; MCC, Austin, USA, 1987.

[Stone75] M. Stonebraker; *Implementation of Integrity Constraints and Views by Query Modification*; Proceedings of the 1975 SIGMOD Conference; San Jose, USA, 1975.

[Tsich82] D.C. Tsichritzis, F.H. Lochovsky; *Data Models*; Prentice-Hall, 1982.

[Wils89] A.N. Wilschut, P.W.P.J. Grefen, P.M.G. Apers, M.L. Kersten; *Implementing PRISMA/DB in an OOPL*; Proceedings of the 6th International Workshop on Database Machines; Deauville, France, 1989.

[Wils90a] A.N. Wilschut, P.W.P.J. Grefen; *PRISMA/DB1 XRA Definition*; PRISMA Document P465; University of Twente, The Netherlands, 1990.

[Wils90b] A.N. Wilschut, P.M.G. Apers; *Pipelining in Query Execution*; Proceedings of the ParBase'90 Conference; Miami Beach, USA, 1990.

[Zloof78] M.M. Zloof; *Security and Integrity within the Query-by-Example Database Management Language*; IBM RC 6982; Yorktown Hts., USA, 1978.