Parallel Handling of Integrity Constraints on Fragmented Relations

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

Abstract

Integrity constraint handling is considered an important issue in relational database management systems. Many studies were already conducted in this area. Little attention has been paid however to the influence of relation fragmentation and parallelism on constraint handling. This paper shows how relation fragmentation complicates matters on the one hand, but how parallelism can help to get better efficiency in enforcing constraints on the other hand. The ideas as presented in this paper are used in the context of the PRISMA database machine; they have a more general applicability though.

1 Introduction

Integrity constraints are an essential part of a data model. Especially in the relational model they are necessary to model the semantics of the applications; the growing complexity of modern applications further increases the need for powerful constraint handling mechanisms. There has been much research in this field already for centralized and traditional distributed database systems, e.g. [Ston75,Simon85,Simon87,Morg84]. Nowadays complex applications for database systems have led to high requirements in the field of system performance. One of the ways to deal with this requirement is the use of relation fragmentation and parallelism in multiprocessor database systems. A large amount of research has been devoted to this issue, e.g. [Cope88,DeWi88,Brat89]; this research focuses mainly on the parallel execution of queries. Little attention has been paid however, to the combination of a powerful integrity enforcement mechanism on the one hand and fragmentation and parallelism on the other hand. 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 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)".

In the PRISMA project [Kers87,Apers88], a parallel relational database machine is developed, that supports integrity constraints. The research in the field of integrity constraints focuses on the problems that relation fragmentation brings to constraint handling in general, and the use of parallelism for improving the efficiency of constraint enforcement within the context of the PRISMA DBMS. Further, attention is paid to a modular design of the constraint handling subsystem to obtain flexibility and extendibility.

This paper is structured as follows. The remainder of this introduction gives a short introduction to the important aspects of the PRISMA DBMS and to the notation and terminology used in this paper. In Section 2 we will show how integrity constraints formulated in terms of global relations can be translated into a fragmented form. Section 3 discusses the strategy for constraint enforcement at a conceptual level. The implementation issues for this strategy are discussed in Section 4. The paper ends with a conclusion and a look into the future of the project with respect to integrity constraints.

This paper does not intend to give a full description of all algorithms used in our approach to integrity constraints, but it rather aims at highlighting the interesting aspects of the integrity constraint handling mechanism and giving the reader a good flavour of the underlying ideas and concepts. To illustrate these concepts we will use two important and well known examples of integrity constraints; the proposed algorithms are applicable to a much larger class of constraints though. A more elaborate description of our approach can be found in [Gref89].

1.1 The context : PRISMA/DB

PRISMA/DB is a parallel, main memory relational database management system, employing horizontal relation fragmentation [Apers88,Kers87]. The system is designed to run on a shared-nothing multi-processor [Bron87]. Figure 1 shows a simplified architecture of PRISMA/DB; parts of the system that are not

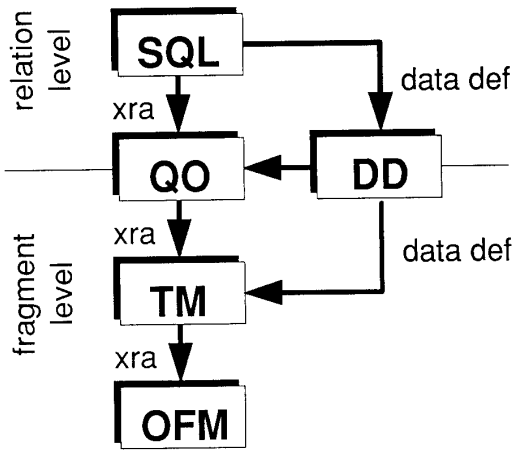

Figure 1: Simplified Architecture of PRISMA/DB

relevant to this paper are left out. The figure shows the configuration for one user session; most components are replicated for every session.

Within a session, the SQL parser (SQL) takes care of data manipulation and data definition in SQL. Data manipulation statements are translated into the internal relational language of PRISMA/DB, called extended relational algebra (XRA), and sent to the query optimizer. Data definition statements are handled in cooperation with the data dictionary; this includes the definition of integrity constraints. The SQL parser operates on relations; fragmentation is transparent here.

The query optimizer (QO) translates queries formulated in terms of global relations into queries formulated in terms of fragments. As such, the QO is the interface between relation and fragment level in the DBMS for data manipulation operations. Further, it removes views and optimizes queries. Output XRA is sent to the transaction manager (TM) for execution.

The data dictionary (DD) stores all central system information, including relation and fragment descriptions. Integrity constraint definitions received from the SQL parser are translated from the relation to the fragment level and optimized, as will be shown later in this paper. The DD is thus the interface between relation and fragment level for data definition operations. The DD uses special purpose data structures for efficient access to its contents.

The transaction manager (TM) takes care of the execution control of a transaction; the TM operates fully at the fragment level of the system. It ensures transaction serializability, atomicity and correctness. For the latter it makes use of information about integrity constraints in the data dictionary. The TM manages the dynamic infrastructure created for the execution of specific relational operations.

The one-fragment manager (OFM) stores fragment data and performs relational operations on it. OFMs can be created and disposed of dynamically; the transaction manager controls this process according to the requirements of the transaction to be executed. OFMs come in two kinds: the first kind is used for the storage of permanent data (base OFMs), the second kind is used for the processing and storage of intermediate results of queries.

As lined out above and depicted in figure 1, there is a clear separation between the relation and fragment level in PRISMA/DB.

1.2 Notation formalism

Given a database schema $\mathcal{D} =< \mathcal{R}_1, \cdots, \mathcal{R}_n >$, we use the following notation:

- D of type $\mathcal{D}$ is a database extension or state of $\mathcal{D}$;
- R of type $\mathcal{R}$ is a relation extension or state of $\mathcal{R}$;
- $\mathcal{D}^* =< \pi_1\mathcal{R}_1, \cdots, \pi_n\mathcal{R}_n >$ is a partial database schema; π_i is a simple projection on the type of a relation;
- D^* of type $\mathcal{D}^*$ is a partial state of $\mathcal{D}$.

Now we define an integrity constraint as a pair $I = [t, r]$, with the following elements:

- $t \subseteq T$ is the set of *triggers* of I; T is the set of all triggers, this is the set of all possible combinations of an update operation type and a relation in D;
- r is a function of type $\mathcal{D}^* \rightarrow bool$; r is called the *rule* of I.

The semantics of I is as follows: when one of the triggers of t is activated, rule r is evaluated on state D^* of that moment; if this evaluation results $false$, the transaction is aborted; otherwise, there is no action. The evaluation of the rule and the conditional transaction abort are called the *enforcement* of the constraint.

Notice, that this way of desribing integrity constraints combines a functional description (the rule), with operational semantics (the triggers). This description of constraints is formulated at the relation level of the database; an analogous description with obvious semantics can be used for the fragment level. A comparable approach of explicitly stating triggers and rules in constraint definitions can be found in the QBE language [Zloof78].

The following example expresses a referential integrity constraint from attribute i of relation R, denoted as $R.i$, to attribute j of relation S [Date81]:

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

In the remainder of this paper we shall assume the existence of two relations R and S, both horizontally fragmented into fragments $R_1, \cdots R_m$ resp. $S_1, \cdots, S_n$ for illustration of constraint handling.

2 Translating constraints

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 is necessary. The translation is performed statically, which means at constraint definition

time; this prevents the overhead of translating constraints every time they are to be enforced. Further, the translation needs to be redone in case of a change in the fragmentation of the involved relations. In PRISMA/DB, translated constraints are stored in the data dictionary of the system for use by the transaction manager.

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 three sequential steps:

- translation of the constraint at the relation level into canonical 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.

2.1 Relation level

We will use two examples in this paper: a domain constraint $I1$ and a referential integrity constraint $I2$; an elaboration of a uniqueness constraint can be found in [Gref89]. These examples are illustrative for a much larger class of constraints.
At the relation level the domain constraint is defined as follows:

$$\begin{aligned} I1_{rel} &= [t1_{rel}, r1_{rel}] \\ t1_{rel} &= \{INS(R), UPD(R)\} \\ r2_{rel} &= (\forall x \in R.i)(c(x)) \end{aligned}$$

in which $c(x)$ is some boolean condition over x. The referential integrity constraint definition is:

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

In this definition, $S.j$ is a key (unique attribute) of relation S.

2.2 Canonical form

Translation of a constraint $I = [t, r]$ from relation level to canonical form at the fragment level is accomplished as follows:

- replace each trigger in t defined on relation R by the set of equivalent triggers defined on all fragments of R;
- replace each occurrence of a global relation R in r with the algebraic reconstruction expression of that relation out of its fragments.

This process applied to the two examples defined above results in the following canonical definitions at the fragment level:

$$\begin{aligned} I1_{can} &= [t1_{can}, r1_{can}] \\ t1_{can} &= \{INS(R_1), \ldots, INS(R_m), \\ &\quad UPD(R_1), \cdots, UPD(R_m)\} \\ r1_{can} &= (\forall x \in (R_1.i \cup \cdots \cup R_m.i))(c(x)) \end{aligned}$$

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

2.3 Distributed form

Although the canonical constraint is formulated in terms of fragments, the semantics is still relation oriented: both the trigger set and the rule of the constraint are specified with respect to the reconstruction of global relations. To obtain fragment-oriented semantics, the canonical form is translated into the so-called distributed form. To obtain this form, the following two steps are made:

- replace the canonical constraint by a set of *local* (fragment) constraints; the number of local constraints is equal to the number of fragments used in the trigger set of the constraint; the trigger set of each local constraint is a subset of the trigger set of the canonical constraint, in which each trigger is defined in terms of the same relation fragment; the rule of a local constraint is equal to the rule of the canonical constraint;
- simplify the rule of each local constraint where possible to prevent unnecessary access to other fragments.

To distribute domain constraint $I1$, we replace the canonical constraint with its trigger set defined in terms of m fragments of relation R, by the set of constraints $CS1 = \{I1_{R_k} \mid 1 \leq k \leq m\}$; each constraint in this set has its trigger set defined on only one fragment of R:

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

It is clear, that the definition of the rule of the canonical constraint can be simplified to the above, because the rule can be evaluated on every value in R_k individually. Each constraint $I1_{R_k}$ is thus defined completely in terms of only one fragment of the involved relation; this type of constraint we call *completely distributable.*

Referential integrity constraint $I2$ is defined in terms of two relations R and S. Therefore, $I2$ is replaced by two sets of local constraints $CS2_R = \{I2_{R_k} \mid 1 \leq k \leq m\}$ and $CS2_S = \{I2_{S_k} \mid 1 \leq k \leq n\}$:

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

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

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.

Note that we require that the fragmentation of a relation is always consistent. This means, that updates on fragments may cause tuple migration: tuples that violate the fragmentation constraint of the fragment they reside in, migrate to the fragment they belong to. This migration process can cause the triggering of more local constraints.

2.4 Optimized form

The distributed form of constraints as discussed above is still very inefficient with respect to constraint enforcement in most cases. Therefore, we consider various ways to optimize the distributed form of constraints presented above:

- restriction of the amount of data to be checked;
- applying algebraic manipulation to the rule;
- using knowledge of relation fragmentation to simplify rules.

2.4.1 Restriction of the amount of data

We can restrict the amount of data to be checked by only checking those parts of fragments that have been changed in a relevant way. This method is already described as differential test [Simon85,Simon87,Gard89]. Using differential tests requires the definition of the following two concepts:

- R^+ denotes tuples inserted into R and new values of tuples modified in R;
- R^- denotes tuples deleted from R and old values of tuples modified in R.

As will be clear, domain constraints have to be checked only with respect to new values in the fragments; therefore, we can reformulate our definition of $CS1$ into $CS1_{diff}$:

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

The same technique can be applied to the definition of the referential integrity constraint $CS2$, leading to the differential definition $CS2_{diff}$:

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

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

Note that the form of the rule of $I2_{S_k}$ had to be modified to use the concept of differential tests. Informally, the rule states: all values that have been deleted from the referenced attribute of fragment S_k and that have not been inserted again in any of the fragments of S, may not be in the referencing attribute of any of the fragments of R. The restriction of $R.i$ to non-null values is not strictly necessary in this rule, since $S.j$ (being a key) may not contain null values.

2.4.2 Rule manipulation

In query optimization it is very common to rewrite expressions to optimize the execution of the expression. For fragmented databases, one of the most used techniques is pushing operations through unions to obtain computations that are local to the fragments. A comparable approach can be applied to the rules of integrity constraints. Here we can push a quantifier through a union operator using the following rewriting rules:

$$(\forall x \in \bigcup_{k=1}^{k=m} R_k)(c(x)) \rightarrow \bigwedge_{k=1}^{k=m}((\forall x \in R_k)(c(x)))$$

$$(\exists x \in \bigcup_{k=1}^{k=m} R_k)(c(x)) \rightarrow \bigvee_{k=1}^{k=m}((\exists x \in R_k)(c(x)))$$

Applied to the referential integrity constraint set $CS2_{diff}$ we obtain a form $CS2_{man}$ in which the union of fragments of a relation is removed; we only need to construct a union for differential sets in the definition of $I2_{S_k}$:

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

$$
\begin{aligned}
I2_{S_k} &= [t2_{S_k}, r2_{S_k}] \\
t2_{S_k} &= \{DEL(S_k), UPD(S_k)\} \\
r2_{S_k} &= (\forall y \in (S_k^-.j - (S_1^+.j \cup \cdots \cup S_n^+.j))) \\
&\quad (\bigwedge_{v=1}^{v=m}((\forall x \in R_v.i)(x \neq y)))
\end{aligned}
$$

Notice that the rules above have quantified expressions at the lowest level in the expressions. This means that these computation intensive tasks have been distributed across fragments of a relation, thus giving possibilities for employing parallelism. We will show later, that our approach makes good use of these possibilities.

2.4.3 Usage of fragmentation knowledge

We can use fragmentation knowledge in the optimization of constraints in the same way as this knowledge can be used in query

optimization [Ceri84]. In query optimization, branches of query trees are removed that can never give any result; in constraint optimization, branches in the rule expression are removed that are known to be always satisfied. The following example illustrates this.

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 $CS2_{diff}$ to the following:

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

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

It is clear, that enforcement of this constraint is much easier (and cheaper) than the enforcement $CS2_{diff}$. This leads to the observation, that integrity constraints should be taken into account in relation fragmentation design.

3 Enforcing constraints

In the previous section we have shown how constraints specified at the relation level, can be translated to the fragment level and manipulated to obtain a form that is fit for straightforward implementation of constraint enforcement algorithms. In this section we will show how these constraint specifications can indeed be mapped into eXtended Relational Algebra (XRA) expressions that implement the rules of the constraints. The next section discusses implementation and execution of these XRA expressions.

The use of XRA as an enforcement vehicle gives several important advantages over specialized *ad hoc* algorithms:

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

The remainder of this section first discusses the most important XRA operators for describing integrity enforcement algorithms. Next, attention is paid to the mapping of constraint definitions onto XRA expressions.

3.1 XRA constructs

For constraint enforcement in XRA, we will 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 used for normal query processing in PRISMA/DB.

We have two operators in XRA 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 will be shown in the next section, 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:

$$alarm(oper)$$

3.2 Mapping constraints to XRA

In this section we will show how constraint definitions as discussed in the previous section can be mapped to XRA expressions. It is not our goal to give rules that describe this mapping for arbitrary constraints. Instead we will show the process for some representative cases. In these cases we will use the transformation rules as shown below; these rules show how a specific logic construct as appearing in the constraint definitions can be mapped onto an XRA construct.

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

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

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

$$(\forall x \in R)(\forall y \in S)(x \neq y) \rightarrow \quad (4)$$
$$alarm(R \cap S)$$

$$(\forall x \in R)(\bigwedge_{i=1}^{n}(\forall y \in S_i)(x \neq y)) \rightarrow \quad (5)$$
$$copy(R, T_1, \cdots, T_n)$$
$$alarm(T_1 \cap S_1)$$
$$\vdots$$
$$alarm(T_n \cap S_n)$$

Domain constraint $CS1_{diff}$ 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 the first case of referential integrity constraint $CS2_{man}$ as discussed in section 2.4.2 as a second example:

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

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

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

Note, that the *copy* operator takes care of ditributing the differential set of R_k; this is used to obtain pipeling parallelism, as discussed in the next section. If we know that relation S is fragmented on attribute $S.j$ using fragmentation constraints $fc_1, \cdots, fc_n$, we can change the XRA construct above as follows:

$$split(unique(\pi_i(\sigma_{i \neq null}(R_k^+))), temp_1, fc_1, \cdots, temp_n, fc_n)$$
$$alarm(temp_1 - \pi_j(S_1))$$
$$\vdots$$
$$alarm(temp_n - \pi_j(S_n))$$

The second case of $CS2_{man}$ can be mapped onto XRA in a comparable way using transformation 5; the constraint definition is the following:

$$\begin{aligned} I2_{S_k} &= [t2_{S_k}, r2_{S_k}] \\ t2_{S_k} &= \{DEL(S_k), UPD(S_k)\} \\ r2_{S_k} &= (\forall y \in (S_k^-.j - (S_1^+.j \cup \cdots \cup S_n^+.j))) \\ &\quad (\bigwedge_{v=1}^{v=m}((\forall x \in R_v.i)(x \neq y))) \end{aligned}$$

The matching XRA construct is:

$$copy((\pi_j(S_k^-) - (\pi_j(S_1^+) \cup \cdots \cup \pi_j(S_n^+))), temp_1, \cdots, temp_m)$$
$$alarm(temp_1 \cap \pi_i(R_1))$$
$$\vdots$$
$$alarm(temp_m \cap \pi_i(R_m))$$

4 Implementation issues

The XRA constructs used for 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.

4.1 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;
- temporary OFM : used for execution of relational operators
- 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 logic to automaticaly 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 dynamicaly 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 execution [Wils89,Wils90].

To obtain a complete transaction model, 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 equaly well. Enforcing constraints at the end of a transaction consists 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 [Wils90].

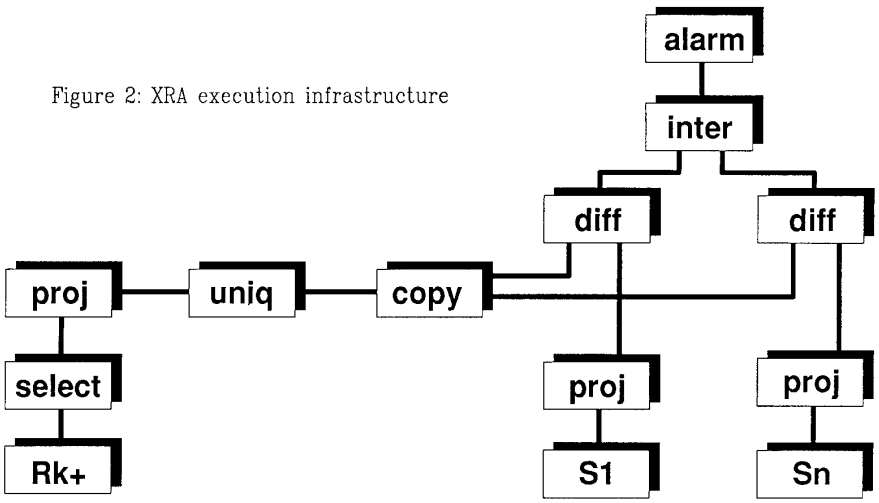

Figure 2: XRA execution infrastructure

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 (TM-OFM). 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 the insert case of referential integrity rule $CS2_{man}$:

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

can straightforwardly be implemented by the execution infrastructure as shown in Figure 2.

If several fragments of relation R have to be checked, it is not necessary to duplicate the complete infrastructure; this would lead to excessive overhead if the fragmentation degree is high. The same infrastructure can be used, if Rk^+ is replaced by the union of all fragments involved; in this way, sharing of resources is used to be able to control the overhead involved.

4.2 Parallelism in constraint enforcement

We have seen that constraint emforcement is executed by a XRA infrastructure. The relational operators in these infrastructures are all independent processes, so parallelism can be employed easily. Within the constraint enforcement mechanism we can distinguish three types of parallelism [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 enforcement level, we can consider this a kind 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 2 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.

Because PRISMA/DB is designed to run on a multi-processor with many nodes (the prototype has 100 nodes), parallelism at the logical level can easily be made operational by assigning OFMs to nodes in the system. We expect that the use of parallelism will reduce response times in constraint enforcements strongly, thereby taking away one of the largest obstacles for the widespread use of constraints in relational databases.

5 Conclusion and Look into the Future

In this paper we have given an outline of a way to combine integrity constraint handling on the one hand and relation fragmentation and parallelism on the other hand in a relational DBMS. The way constraints are handled in PRISMA/DB is of a general and modular structure, thereby giving easy ways for modification and extention. The constraint handling mechanism in PRISMA/DB consists of two subsystems:

- a translation mechanism that translates constraints from the relation level to the fragment level and optimizes these; this can be seen as the static part of the subsystem; this mechanism is located in the data dictionary of the DBMS;
- an enforcement mechanism that enforces the constraints as produced by the translation mechanism using XRA execution infrastructures; this is the dynamic part of the subsys-

tem; this mechanism is located in the transaction manager of the DBMS.

Although relation fragmentation complicates constraint handling, it also allows for a high degree of parallelism in the enforcement algorithms, thereby giving good possibilities to reduce response times, comparable to the gains obtained by using parallelism in normal query execution.

We think that the constraint handling mechanism as proposed in this paper contributes to the use of integrity constraint handling in parallel database systems; the proposal both addresses the complications of relation fragmentation and the benefits of parallelism. To the best of our knowledge, there have not been any proposals that cover this topic.

A prototype implementation of PRISMA/DB is operational now. We plan to enhance this prototype with integrity constraint handling as lined out in this paper. This prototype will support at least the structural constraint classes of the relational model [Gard89]:

- domain constraints
- nonnull constraints
- uniqueness constraints
- referential integrity constraints

Using this prototype experiments will be conducted to measure the response time benefits of using parallelism in constraint enforcement.

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, 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.

[Brat89] K.Bratbergsengen, T.Gjelsvik; *The Development of the CROSS8 and HC16-186 Parallel (Database) Computers*; Proceedings of the 6th International Workshop on Database Machines; Deauville, France, 1989;

[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.

[Cope88] G.Copeland et al.; *Data Placement in Bubba*; Proceedings of the 1988 SIGMOD Conference; Chicago, USA, 1988.

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

[DeWi88] D.J.DeWitt et al.; *A Performance Analysis of the Gamma Database Machine*; Proceedings of the 1988 SIGMOD Conference; Chicago, USA, 1988.

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

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

[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.

[Ston75] M.Stonebraker; *Implementation of Integrity Constraints and Views by Query Mofification*; Proceedings of the 1975 SIGMOD Conference; San Jose, USA, 1975.

[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.

[Wils90] 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.