Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

Managing Global Constraints in Multidatabase Systems

Chih-Shyang Chang
Trilogy Technologies, Inc.
3F, No. 3, Lane 181, Section 2,
Anho Road, Taipei, Taiwan, R.O.C.
Email: hank@trilogy.com.tw
Fax: 02-377-0594

Arbee L.P. Chen*
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.
Email: alpchen@cs.nthu.edu.tw
Fax: 03-5723694

Abstract

Various issues in multidatabase systems were studied, including resolution of interoperability problems, database integration, query processing, and transaction management. However, deriving global constraints *from the individual local constraints is rarely considered. Global constraints in multidatabase systems may be used to (a) avoid local constraint checking, (b) accomplish semantic query optimization, (c) resolve interoperability problems, and (d) specify semantic relationships among databases. In this paper, we analyze the types of local constraints and the ways of global schema construction to come up with a set of rules for global constraint derivation. The enforcement of the global constraints considering various types of update operations is also presented.*

1 Introduction

A closed multidatabase system [14] provides the facility by which users can make requests to a global schema. To support the global schema, schema integration from a set of local autonomous databases is necessary [8, 13, 16]. Because local databases are developed independently with different requirements, a multidatabase system is likely to have many similar objects with different structures or representations. As a result, various conflicts among the local databases exist, which causes the interoperability problems [2, 3, 14]. A query against the global schema is decomposed into a set of subqueries — one for each local DBMS that will be involved in query execution. The optimization of global query processing and the integration of local schemas have been broadly studied [4, 5, 20, 21]. However, global constraint management is rarely considered in multidatabase systems.

There are some works on global constraint enforcement in distributed database systems. The work of Tanaka and Kambayashi [19] discussed the constraint (i.e., functional, join, and embedded join dependencies) preservation problem in a global view constructed by a join operator. Qian [17] proposed a top-down constraint reformulation approach for the constraints, which can only be applied in distributed database systems. Gupta and Widom [10] presented a method to verify constraints by accessing remote data in distributed databases. Both works [10, 17] attempted to optimize the constraint checking cost in distributed database systems. The work of Reddy, *et al.* [18] considered deriving new constraints that offer the potential for reducing query processing cost in multidatabase systems. However, the enforcement of constraints was not considered. In this paper, we propose an approach to derive global constraints in multidatabase systems, and discuss their enforcement strategies. In the following, we list some advantages of supporting global constraints in multidatabase systems.

- The global constraints may avoid local constraint checking. That is, the global constraints can reject an update before it is sent to local sites for processing. In this case, the users know which global constraints are violated by the update. Also, the constraint enforcement is efficient.
- The global constraints can be used for semantic query optimization [12, 18, 22].
- The global constraints may provide useful information for the resolution of semantic heterogeneities in multidatabase systems [9].
- New constraints may be derived, which do not exist in the individual local databases [18].

The rest of the paper is organized as follows. Section 2 describes our approach, and introduces the categories of the constraints and the relationships between the predicates on a view and in a constraint. Section 3 describes the derivation of the constraints for different views constructed using relational operations. Section 4 discusses the enforcement of the global constraints. We conclude the paper in Section 5. The examples used in this paper are based on the following company schema:

$$EMP(name, dept, age, salary)$$
$$DEPT(dept, budget, chairman)$$
$$PROJ(proj, dept, manager)$$
$$ASSIGN(name, proj)$$

Figure 1: A company schema.

where the attributes *manager*, *chairman*, and *name* have the same domain. Also, the attributes with the same name in different schemes are implicitly associated with the same domain.

2 The Approach

A relation scheme in a global schema can be considered as a view. The views can be defined by the relational operations [7, 11, 15]. We take selection, projection, difference, intersection, join, union, Cartesian product, and outer-join

*To whom all correspondence should be sent.

as the operations to define a view [7, 11, 15]. It is undecidable to determine whether an arbitrary view, which can be constructed by the base relations and/or other views, is updatable or not [7, 11, 15]. In this paper, we consider the updatable views which are constructed from only the base relation(s) via the relational operations. Also, when we say "a tuple in a view" in the following, we mean "a tuple in the view when it is materialized."

There are two possible derivations of the global constraints from local constraints. For example, a view $\mathcal{V}$ which is constructed by the relational operation **intersection** on two local relations R_1 and R_2 is denoted by $\mathcal{V} = R_1 \cap R_2$, where relations R_1 and R_2 have the same attribute set. Let C_1, $\{\forall x \in R_1 : 70 > x.A > 30\}$, and C_2, $\{\forall x \in R_2 : 65 > x.A > 25\}$, be two constraints on R_1 and R_2, respectively. Consider the derivation of the global constraints on $\mathcal{V}$ from C_1 and C_2. One way to specify the constraint on A in $\mathcal{V}$ is $\{\forall x \in \mathcal{V} : 65 > x.A > 30\}$. Another way can be $\{\forall x \in \mathcal{V} : 70 > x.A > 25\}$. The global constraint derived by the first way is called a *strong* global constraint while that by the second way a *weak* global constraint. In this paper, we consider only the derivations of the strong global constraints.

A global constraint is strong if it requires that the tuples which satisfy the global constraint satisfy the corresponding individual local constraint(s). That is, if a tuple satisfies a strong global constraint, then the corresponding tuples in the base relation(s) will not violate any local constraint. However, there exist local constraints from which a strong global constraints cannot be derived.

The enforcement of the global constraints is also considered in this paper. To enforce a global constraint, local data accesses may be needed depending on the types of the constraints. We categorize the constraints and provide the constraint enforcement strategies when local data accesses are needed.

The work we consider in this paper can be summarized as follows:

> Given a view $\mathcal{V}$ constructed from the base relations with constraints C's, derive a strong global constraint $\mathcal{VC}$ from C's on $\mathcal{V}$ and provide constraint enforcement strategies for $\mathcal{VC}$.

2.1 Relationships Between Predicates

A *simple predicate* is denoted $x\theta y$, where x is an attribute of a relation (same or different), y can be an attribute of a relation or a constant, and θ is a comparator, $\theta \in \{=, \neq, <, >, \leq, \geq\}$. A *composite predicate* consists of a conjunction of more than one simple predicate. It can be represented as $p_1 \wedge p_2 \wedge \cdots \wedge p_m$ ($m > 1$), where $p_1, p_2, ..., p_m$ are simple predicates. In this paper, we denote a simple predicate or a composite predicate as P, Q, S, or T. A function $attr(P, R)$ is used to obtain the attribute(s) of the relation R, which is involved in predicate P.

For any two predicates (simple or composite) P and Q on a relation R, we consider their relationships as follows. If $attr(P, R) \neq attr(Q, R)$, we say P and Q are *irrelevant*, denoted as $P \asymp Q$ or $Q \asymp P$ (e.g., $P = \{EMP.salary \geq 30k\}$ and $Q = \{EMP.age \leq 30\}$). The following relationships are discussed based on the assumption that $attr(P, R) = attr(Q, R)$.

- *equivalent* ($\equiv$)
  P and Q are *equivalent* if for each tuple t of R, t satisfies P if and only if t satisfies Q. It is denoted as $P \equiv Q$ or $Q \equiv P$ (e.g., $P = \{EMP.age = 40\}$ and $Q = \{EMP.age \geq 40 \wedge EMP.age \leq 40\}$).
- *subset* ($\prec$)
  P is a *subset* of Q if for each tuple t of R, if t satisfies P then t satisfies Q. It is denoted as $P \prec Q$ (e.g., $P = \{EMP.age \geq 40\}$ and $Q = \{EMP.age \geq 30\}$).
- *superset* ($\succ$)
  P is a *subset* of Q if for each tuple t of R, if t satisfies P then t satisfies Q. It is denoted as $P \succ Q$
- *overlap* ($\bigcirc$)
  P and Q *overlap* if for each tuple t of R, t satisfies either both P and Q or only P or Q. It is denoted as $P \bigcirc Q$ or $Q \bigcirc P$ (e.g., $P = \{EMP.age \geq 30\}$ and $Q = \{EMP.age \leq 40\}$).
- *disjoint* ($\otimes$)
  P and Q are *disjoint* if no tuples of R satisfy P and Q. It is denoted as $P \otimes Q$ or $Q \otimes P$ (e.g., $P = \{EMP.age \geq 40\}$ and $Q = \{EMP.age \leq 30\}$).

2.2 The Constraint Categorization

In this paper, we use *assertion* to specify a constraint [1]. Constraint and assertion are used interchangeably when it causes no confusion. A constraint can be represented as

$$Q_1 x_1 \in R_1\ Q_2 x_2 \in R_2 \ldots Q_n x_n \in R_n : S \Rightarrow T,$$

where $Q_i, 1 \leq i \leq n$, is a quantifier ($\forall$ or $\exists$), $x_i, 1 \leq i \leq n$, is a tuple variable which denotes a tuple of the relation R_i, and $S \Rightarrow T$ is an implication where S, called *antecedent*, and T, called *consequent*, are predicates. "$S \Rightarrow$" can be omitted if the implication is *true* for any antecedent, e.g., $\{\forall x \in EMP : x.age \geq 20\}$.

According to the numbers of tuple variables and relations of an assertion, we classify assertions which do not involve aggregate functions as follows:

- 1V1R assertions
  The 1V1R assertions involve one tuple variable and one relation. It can be represented as
  $$Qx \in R : S \Rightarrow T.$$
  For example, the constraint "*if an employee's age is larger than 40, his/her salary must be larger than 50k.*" is a 1V1R assertion. It can be represented as $\{\forall x \in EMP : x.age > 40 \Rightarrow x.salary > 50k\}$.
- 2V1R assertions
  The 2V1R assertions involve two tuple variables and one relation. It can be represented as
  $$Q_1 x \in R\ Q_2 y \in R : S \Rightarrow T.$$
  For example, the functional dependency $\{dept \rightarrow manager\}$ on the $PROJ$ relation, which can be represented as
  $\{\forall x \in PROJ\ \forall y \in PROJ : x.dept = y.dept \Rightarrow x.manager = y.manager\}$, is a 2V1R assertion.
- 2V2R assertions
  The 2V2R assertions involve two tuple variables and two relations. It can be represented as
  $$Q_1 x \in R_1\ Q_2 y \in R_2 : S \Rightarrow T.$$
  For example, the referential integrity constraint for the *dept* attribute from relation EMP to $DEPT$, which can be represented as $\{\forall x \in EMP\ \exists y \in DEPT : x.dept = y.dept\}$, is a 2V2R assertion.
- MVMR assertions
  The MVMR assertions involve more than two tuple variables and more than two relations.

In this paper, we discuss the 1V1R, 2V1R and 2V2R assertions.

Let $\mathcal{S} =< \mathcal{R}, \mathcal{C} >$ be a relational schema, where $\mathcal{R}$ and $\mathcal{C}$ denote the set of relations and the set of constraints, respectively. A database state s is *consistent* with $\mathcal{C}$ if and only if all constraints in $\mathcal{C}$ are evaluated as true for state s. A tuple in the consistent database state satisfies all constraints in $\mathcal{C}$. There are two ways for a tuple to satisfy a constraint c, ($c \in \mathcal{C}$).

1. *trivially satisfy:* the antecedent of c is evaluated as false, given the values in the tuples. For example, the tuple (Tony, CS, 35, 42k) in EMP relation trivially satisfies the constraint $\{\forall x \in EMP : x.age > 40 \Rightarrow x.salary > 50k\}$.
2. *nontrivially satisfy:* the antecedent and consequent of c are both evaluated as true, given the values in the tuples. For example, the tuple (Alex, EE, 45, 60k) in EMP relation nontrivially satisfies the constraint $\{\forall x \in EMP : x.age > 40 \Rightarrow x.salary > 50k\}$.

3 Global Constraint Derivation

In this section, we derive the view constraints for the various views according to 1V1R, functional dependency, and referential integrity constraints.

3.1 Selection (σ)

The view constructed by the relational operation selection can be represented by $\mathcal{V} = \sigma_{\mathrm{P}}(R)$, where P is the selection predicate. The following subsections present the derivation of the view constraints on $\mathcal{V}$ considering various constraints on R.

3.1.1 1V1R Assertions

Example 1 Consider the view:

$$\mathcal{V}_1 = \sigma_{dept=\text{"R\&D"}}(EMP)$$

and a 1V1R constraint C_1 on EMP:

$$(C_1) \quad \forall x \in EMP : x.dept = \text{"R\&D"} \Rightarrow x.salary \geq 40k$$

According to constraint C_1, the *salary* attribute values of the tuples in $\mathcal{V}_1$ are all larger than or equal to 40k. That is, all the tuples in $\mathcal{V}_1$ nontrivially satisfy constraint C_1. Hence, the view constraint on $\mathcal{V}_1$ can be derived (replace EMP by $\mathcal{V}_1$ and remove the antecedent of C_1) as:

$$(VC_1) \quad \forall x \in \mathcal{V}_1 : x.salary \geq 40k$$

Consider another view:

$$\mathcal{V}_2 = \sigma_{dept\neq\text{"R\&D"}}(EMP)$$

According to constraint C_1, all the tuples in $\mathcal{V}_2$ trivially satisfy constraint C_1. Hence, no constraint from C_1 needs to be derived for $\mathcal{V}_2$. ■

Example 2 Consider the view:

$$\mathcal{V}_3 = \sigma_{salary\geq 35k}(EMP)$$

and the 1V1R constraint C_1 shown in Example 1. C_1 can be equivalently represented as follows:

$$(C_1') \quad \forall x \in EMP : x.salary < 40k \Rightarrow x.dept \neq \text{"R\&D"}$$

The tuples in $\mathcal{V}_3$, whose salary values are larger than or equal to 35k and less than 40k (i.e., $\{x.salary < 40k \wedge x.salary \geq 35k\} \equiv \{35k \leq x.salary < 40k\}$), nontrivially satisfy constraint C_1'. The other tuples trivially satisfy constraint C_1'. Hence, the constraint of $\mathcal{V}_3$ can be derived as:

$$(VC_3') \quad \forall x \in \mathcal{V}_3 : 35k \leq x.salary < 40k \Rightarrow x.dept \neq \text{"R\&D"}$$

Consider another view:

$$\mathcal{V}_4 = \sigma_{salary\leq 35k}(EMP)$$

All the tuples in $\mathcal{V}_4$ nontrivially satisfy constraint C_1'. Hence, the constraint on $\mathcal{V}_4$ can be derived by just specifying the consequent of constraint C_1'. That is,

$$(VC_4') \quad \forall x \in \mathcal{V}_4 : x.dept \neq \text{"R\&D."} \blacksquare$$

Property 1 Let a selection view be defined as $\mathcal{V} = \sigma_{\mathrm{P}}(R)$. Consider the 1V1R constraints:

$$(C) \quad \forall x \in R : \mathrm{S} \Rightarrow \mathrm{T}$$

The view constraints can be derived by the rules shown in Table 1.

Conditions	Derivation Rules
$\{\mathrm{P} \equiv \mathrm{S}\} \wedge \{\mathrm{P} \asymp \mathrm{T}\}$	$\{\forall x \in \mathcal{V} : \mathrm{T}\}$
$\{\mathrm{P} \prec \mathrm{S}\} \wedge \{\mathrm{P} \asymp \mathrm{T}\}$	$\{\forall x \in \mathcal{V} : \mathrm{T}\}$
$\{\mathrm{P} \succ \mathrm{S}\} \wedge \{\mathrm{P} \asymp \mathrm{T}\}$	$\{\forall x \in \mathcal{V} : \mathrm{S} \Rightarrow \mathrm{T}\}$
$\{\mathrm{P} \bigcirc \mathrm{S}\} \wedge \{\mathrm{P} \asymp \mathrm{T}\}$	$\{\forall x \in \mathcal{V} : \mathrm{P} \wedge \mathrm{S} \Rightarrow \mathrm{T}\}$
$\{\mathrm{P} \otimes \mathrm{S}\} \wedge \{\mathrm{P} \asymp \mathrm{T}\}$	-
$\{\mathrm{P} \asymp \mathrm{S}\} \wedge \{\mathrm{P} \equiv \mathrm{T}\}$	-
$\{\mathrm{P} \asymp \mathrm{S}\} \wedge \{\mathrm{P} \prec \mathrm{T}\}$	-
$\{\mathrm{P} \asymp \mathrm{S}\} \wedge \{\mathrm{P} \succ \mathrm{T}\}$	$\{\forall x \in \mathcal{V} : \mathrm{P} \wedge \neg\mathrm{T} \Rightarrow \neg\mathrm{S}\}$
$\{\mathrm{P} \asymp \mathrm{S}\} \wedge \{\mathrm{P} \bigcirc \mathrm{T}\}$	$\{\forall x \in \mathcal{V} : \mathrm{S} \Rightarrow \mathrm{T}\}$
$\{\mathrm{P} \asymp \mathrm{S}\} \wedge \{\mathrm{P} \otimes \mathrm{T}\}$	$\{\forall x \in \mathcal{V} : \neg\mathrm{S}\}$
$\{\mathrm{P} \asymp \mathrm{S}\} \wedge \{\mathrm{P} \asymp \mathrm{T}\}$	-

Table 1: The 1V1R constraint derivation rules for selection views.

The "-" shown in Table 1 means that no constraint from C needs to be derived.

3.1.2 2V1R Assertions

Consider a 2V1R constraint, using functional dependency (FD) as an example. The instances of the relation $PROJ$ are shown in Figure 2. The functional dependency $\{dept \rightarrow manager\}$

	proj	*dept*	*manager*
1	NSC92	CS	Tony
2	NSC93	CS	Tony
3	ITRI92	EE	Alex
4	ITRI93	IE	Joe

Figure 2: The instances of the $PROJ$ relation.

holds on the $PROJ$ relation. It can be represented as $\forall x \in PROJ\ \forall y \in PROJ : x.dept = y.dept \Rightarrow x.manager = y.manager$.

Example 3 A view $\mathcal{V}_5$ is defined as follows:

$$\mathcal{V}_5 = \sigma_{dept=\text{"CS"}}(PROJ)$$

The view contains tuples 1 and 2 in Figure 2. If a tuple to be inserted into $\mathcal{V}_5$ satisfies $\{dept \rightarrow manager\}$ in $\mathcal{V}_5$ then it will satisfy the FD in the relation $PROJ$. This is because the attribute involved in the selection predicate of the is the same as that of the left-hand side of the FD.

Consider a view whose selection predicate does not involve the same attribute as the left-hand side of the functional dependency $\{dept \rightarrow manager\}$. The view is defined as:

$$\mathcal{V}_6 = \sigma_{manager\leq\text{"Joe"}}(PROJ)$$

$\mathcal{V}_6$ contains tuples 3 and 4. To insert a tuple (NSC94, CS, Frank) into $\mathcal{V}_6$ is not allowed since it will violate the functional dependency $\{dept \rightarrow manager\}$ in the $PROJ$ relation. although it satisfies the functional dependency $\{dept \rightarrow manager\}$ in $\mathcal{V}_6$. ■

From the above example, we have the Property 2.

Property 2 Let a selection view be defined as $\mathcal{V} = \sigma_{\mathrm{P}}(R)$ and a functional dependency on relation R be $A \rightarrow B$, where A, B are attribute sets of R. The view constraint can be derived by the rules shown in Table 2.

Conditions	Derivation Rules
$attr(\mathrm{P},R) = A$	$\{A \rightarrow B\}$
$attr(\mathrm{P},R) \neq A$	can not derive

Table 2: The FD constraint derivation rules for selection views.

3.1.3 2V2R Assertions

Let a referential integrity constraint, RI, on the company schema as shown in Figure 1 be defined as:

$$(RI) \quad \forall x \in ASSIGN \; \exists y \in PROJ : x.proj = y.proj$$

The instances of the relation $ASSIGN$ are given in Figure 3.

	name	*proj*
1	John	ITRI93
2	Mary	ITRI92
3	Tom	ITRI92

Figure 3: The instances of the $ASSIGN$ relation.

Example 4 Let $\mathcal{V}_7$ and $\mathcal{V}_8$ be two selection views on $ASSIGN$ and $PROJ$ relations, respectively, defined as follows: $\mathcal{V}_7 = \sigma_{proj \leq \text{"ITRI92"}}(ASSIGN)$ and $\mathcal{V}_8 = \sigma_{proj \leq \text{"ITRI93"}}(PROJ)$ $\mathcal{V}_7$ contains tuples 2 and 3 in Figure 3 and $\mathcal{V}_8$ contains tuples 3 and 4 in Figure 2.

Consider the referential integrity constraint RI. To derive the corresponding referential integrity constraint on $\mathcal{V}_7$ and $\mathcal{V}_8$, we consider constraint VRI as follows:

$$(VRI) \quad \forall x \in \mathcal{V}_7 \; \exists y \in \mathcal{V}_8 : x.proj = y.proj$$

If the tuple (ITRI93, IE, Joe) is deleted from $\mathcal{V}_8$, since ITRI93 is not in $\mathcal{V}_7$, constraint VRI is satisfied. However, the deletion will violate RI since the tuple (John, ITRI93) exists in the relation $ASSIGN$. We have Property 3. ■

Property 3 Let a referential integrity constraint on relations R_1 and R_2 be defined as:

$$\forall x \in R_1 \; \exists y \in R_2 : x.A = y.A$$

For the selection views on R_1 and R_2, $\mathcal{V}_1 = \sigma_{\mathrm{P}_1}(R_1)$ and $\mathcal{V}_2 = \sigma_{\mathrm{P}_2}(R_2)$, respectively, the view constraints can be derived by the rules shown in Table 3.

Conditions	Derivation Rules
$\mathrm{P}_1 \equiv \mathrm{P}_2$	$\forall x \in \mathcal{V}_1 \; \exists y \in \mathcal{V}_2 : x.A = y.A$
$\mathrm{P}_1 \prec \mathrm{P}_2$	$\forall x \in \mathcal{V}_1 \; \exists y \in \mathcal{V}_2 : x.A = y.A$
otherwise	can not derive

Table 3: The referential integrity constraint derivation rules for selection views.

3.2 Projection (Π)

The view constructed by the relational operation projection can be defined as $\mathcal{V} = \Pi_A(R)$, where A is a set of attributes of the relation R. Consider an example of 1V1R constraint on R:

$$\forall x \in R : \mathrm{S} \Rightarrow \mathrm{T}$$

If $attr(\mathrm{S},R) \cup attr(\mathrm{T},R) \subseteq A$, the corresponding view constraint can be derived by replacing R with $\mathcal{V}$; otherwise, the view constraint can not be derived. For example, consider the constraint C_1 in Example 1 and the view $\mathcal{V}_9 = \Pi_{name,salary}(EMP)$. The corresponding view constraint of C_1 can not be derived because C_1 specifies the relationships between the *dept* and *salary* attribute values of the tuples. $\mathcal{V}_9$ contains the *name* and *salary* attributes. It does not involve the *dept* and *salary* attributes. Hence, we have the Property 4.

Property 4 Consider the projection views:

$$\mathcal{V}_i = \Pi_{A_i}(R_i), i = 1, ..., m,$$

where A_i is the attribute(s) of the relation R_i. Consider the constraint

$$(C) \quad Q_1 x_1 \in R_1 \; Q_2 x_2 \in R_2 \; ... \; Q_m x_m \in R_m : \mathrm{S} \Rightarrow \mathrm{T}.$$

The view constraint can be derived by the rules shown in Table 4.

Conditions	Derivation Rules
$\cup_{j=1}^{m} attr(\mathrm{S},R_j) \cup attr(\mathrm{T},R_j) \subseteq \cup_{j=1}^{m} A_j$	$R_i \leftarrow \mathcal{V}_i, 1 \leq i \leq m$
otherwise	can not derive

Table 4: The constraint derivation rules for projection views.

"$R_i \leftarrow \mathcal{V}_i$" shown in Table 4 denotes that R_i is replaced by $\mathcal{V}_i$ in constraint C.

3.3 Difference ($-$)

The view constructed by the relational operation difference can be defined as $\mathcal{V} = R_x - R_y$, where the relations R_x and R_y have the same attribute set. The difference views consist of the tuples which are in R_x but not in R_y. The views are updatable, if (a) the tuple which will be inserted into $\mathcal{V}$ does not exist in R_y and (b) the deletion from $\mathcal{V}$ must only be translated to the deletion of R_x. Hence, we can consider the view to consist of only the subset of tuples of R_x. The view constraint derivations is just to replace R_x by $\mathcal{V}$ in any constraint. The derivation rules are described in Table 5.

Property 5 Consider the difference view, defined as $\mathcal{V} = R_x - R_y$, where the relations R_x and R_y have the same attribute set. Consider the constraint C:

$$(C) \quad Q_1 x_1 \in R_1 \; Q_2 x_2 \in R_2 \; ... \; Q_m x_m \in R_m : \mathrm{S} \Rightarrow \mathrm{T}$$

The view constraint can be derived by the rules shown in Table 5.

3.4 Intersection ($\bigcap$)

The view constructed by the relational operation intersection can be defined as $\mathcal{V} = R_1 \cap R_2$, where the relations R_1 and R_2 have the same attribute set. The views consist of the identical tuples in the two relations. The effect of the update to the view will be simultaneously translated to the two base relations R_1 and R_2. That is, the tuples in $\mathcal{V}$ satisfy the constraints on R_1 and R_2.

Conditions	Derivation Rules
$R_i = R_x$	$R_i \leftarrow \mathcal{V}$, $1 \leq i \leq m$
otherwise	-

Table 5: The constraint derivation rules for **difference** views.

3.4.1 1V1R Assertions

From the definition of the intersection view, the 1V1R view constraints can be derived by the rules shown in Table 6.

Property 6 Let $\mathcal{V} = R_1 \cap R_2$. C_1 and C_2, shown as follows, are the 1V1R constraints on the relations R_1 and R_2, respectively.

(C_1) $\forall x \in R_1 : \mathrm{S}_1 \Rightarrow \mathrm{T}_1$

(C_2) $\forall x \in R_2 : \mathrm{S}_2 \Rightarrow \mathrm{T}_2$

Conditions	Derivation Rules
	$\{\forall x \in \mathcal{V} : \mathrm{S}_1 \Rightarrow \mathrm{T}_1\} \wedge \{\forall x \in \mathcal{V} : \mathrm{S}_2 \Rightarrow \mathrm{T}_2\}$

Table 6: The 1V1R constraint derivation rules for **intersection** views.

Notice that the blank condition part of Table 6 means that the derivation rules are applied in any condition.

3.4.2 2V1R Assertions

Let the attribute set of R_1 and R_2 be $\{A, B, C, D\}$. Suppose that the functional dependency $\{C \rightarrow D\}$ simultaneously holds on R_1 and R_2. When an inserted tuple in $\mathcal{V}$ which satisfies $\{C \rightarrow D\}$ is translated to the insertion in R_1 and R_2, the tuple may violate the constraint. Thus, the functional dependent constraint can not be derived.

Property 7 Let $\mathcal{V} = R_1 \cap R_2$ be a view, where R_1 and R_2 are the relations with the same attribute set. In any condition, we can not derive the corresponding FD from the same FD's on R_1 and R_2.

3.4.3 2V2R Assertions

Consider two referential integrity constraints RI_1 and RI_2 as shown below:

(RI_1) $\forall x \in X_1 \; \exists y \in Y_1 : x.B_1 = y.B_1$

(RI_2) $\forall x \in X_2 \; \exists y \in Y_2 : x.B_2 = y.B_2$

Consider the **intersection** view $\mathcal{V}$ again. If $X_1 = R_1$, $X_2 = R_2$, and $Y_1, Y_2 \notin \{R_1, R_2\}$, we can derive the corresponding view constraint. This is because if a tuple is inserted into $\mathcal{V}$, we can check its B_1 and B_2 attribute values to see whether they also exist in Y_1's B_1 and Y_2's B_2 attribute values, respectively (if a tuple is deleted from $\mathcal{V}$, no corresponding constraints need to be checked). If (a) $X_1 = R_1$ and $Y_1 = R_2$, or (b) $X_1 = R_2$ and $Y_1 = R_1$, the referential integrity constraint does not exist, because they are merged into the single relation $\mathcal{V}$. In the other condition (i.e., $Y_1 = R_1$ or $Y_2 = R_2$), the corresponding view constraint can not be derived, because the tuples in $R_1 - \mathcal{V}$ may have the same attribute values of B_1. If a tuple is to be deleted from $\mathcal{V}$, we can not decide whether the tuple can be deleted when the tuple is translated to the base relations R_1 and R_2. The derivation rules are shown in Table 7.

Property 8 Let the **intersection** view be defined as $\mathcal{V} = R_1 \cap R_2$. There are two referential integrity constraints RI_1 and RI_2 as above. The corresponding view constraint can be derived by the rules shown in Table 7.

Conditions	Derivation Rules
$X_1 = R_1 \wedge X_2 = R_2$ $Y_1, Y_2 \notin \{R_1, R_2\}$	$\{\forall x \in \mathcal{V} \; \exists y \in Y_1 : x.B_1 = y.B_1\} \wedge$ $\{\forall x \in \mathcal{V} \; \exists y \in Y_2 : x.B_2 = y.B_2\}$
$\{X_1 = R_1 \wedge Y_1 = R_2\} \vee$ $\{X_1 = R_2 \wedge Y_1 = R_1\}$	-
$\{X_2 = R_1 \wedge Y_2 = R_2\} \vee$ $\{X_2 = R_2 \wedge Y_2 = R_1\}$	-
otherwise	can not derive

Table 7: The 2V2R constraint derivation rules for **intersection** views.

3.5 Join ($\bowtie$)

A view defined as the (natural) **join** operation of the relations R_1 and R_2 can be represented by $\mathcal{V} = R_1 \bowtie R_2$. We discuss the view constraint derivation as follows.

3.5.1 1V1R Assertions

From the definition of the **join** view, the 1V1R view constraints can be derived by the rules shown in Table 8.

Property 9 Let $\mathcal{V} = R_1 \bowtie R_2$. C_1 and C_2 shown as follows are the 1V1R constraints on relation R_1 and R_2, respectively.

(C_1) $\forall x \in R_1 : \mathrm{S}_1 \Rightarrow \mathrm{T}_1$

(C_2) $\forall x \in R_2 : \mathrm{S}_2 \Rightarrow \mathrm{T}_2$

Conditions	Derivation Rules
	$\{\forall x \in \mathcal{V} : \mathrm{S}_1 \Rightarrow \mathrm{T}_1\} \wedge \{\forall x \in \mathcal{V} : \mathrm{S}_2 \Rightarrow \mathrm{T}_2\}$

Table 8: The 1V1R constraint derivation rules for **join** views.

3.5.2 2V1R Assertions

Consider the following example.

Example 5 Let $\{A, B, C\}$ and $\{A, D, E\}$ be the attribute sets of the relations R_1 and R_2, respectively. Consider the instances of the relations R_1 and R_2 shown in Figure 4. suppose that R_1 holds on the functional dependency $\{B \rightarrow C\}$ and R_2 holds on the functional dependency $\{D \rightarrow E\}$. The instances of $\mathcal{V}$ are shown in Figure 5. We know that $\mathcal{V}$ satisfies the functional dependency $\{B \rightarrow C\}$ and $\{D \rightarrow E\}$. Suppose that the tuple $(a_3, b_2, c_2, d_3, e_3)$ is to be inserted into $\mathcal{V}$. According to Figure 5, this tuple will hold on the functional dependency $\{B \rightarrow C\}$ and $\{D \rightarrow E\}$. However, when the tuple is translated to the insertion in R_1 and R_2, it violates $\{B \rightarrow C\}$ in R_1 and $\{D \rightarrow E\}$ in R_2. In this case, the view constraints can not be derived. ■

Property 10 Let $\mathcal{V} = R_1 \bowtie R_2$ be a **join** view. In any condition, we can not derive the corresponding FD from the FD's on R_1 and R_2.

R_1

A	B	C
a_1	b_1	c_1
a_2	b_1	c_1
a_3	b_2	c_1

R_2

A	D	E
a_1	d_1	e_1
a_2	d_2	e_2
a_4	d_3	e_2

Figure 4: The instances of the relations R_1 and R_2.

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_1	c_1	d_2	e_2

Figure 5: The instances of $\mathcal{V} = R_1 \bowtie R_2$.

3.5.3 2V2R Assertions

The derivation of the view constraints of the **join** views are similar to that of the **intersection** views for referential integrity constraints. The derivation rules are shown in Table 9.

Property 11 Let $\mathcal{V} = R_1 \bowtie R_2$ be a **join** view. There are two referential integrity constraints RI_1 and RI_2 as shown below:

(RI_1) $\forall x \in X_1\ \exists y \in Y_1 : x.B_1 = y.B_1$

(RI_2) $\forall x \in X_2\ \exists y \in Y_2 : x.B_2 = y.B_2$

The corresponding view constraint can be derived by the rules shown in Table 9.

Conditions	Derivation Rules
$X_1 = R_1 \wedge X_2 = R_2$ $Y_1, Y_2 \notin \{R_1, R_2\}$	$\{\forall x \in \mathcal{V}\ \exists y \in Y_1 : x.B_1 = y.B_1\} \wedge$ $\{\forall x \in \mathcal{V}\ \exists y \in Y_2 : x.B_2 = y.B_2\}$
$X_1 = X_2 = R_1 \wedge$ $Y_1 = Y_2 = R_2 \wedge$ $B_1 = B_2 = A$	-
$X_1 = X_2 = R_2 \wedge$ $Y_1 = Y_2 = R_1 \wedge$ $B_1 = B_2 = A$	-
otherwise	can not derive

Table 9: The 2V2R constraint derivation rules for **join** views.

3.6 Union ($\bigcup$)

The view constructed by the relational operation **union** can be defined as $\mathcal{V} = R_1 \cup R_2$, where the relations R_1 and R_2 have the same attribute set. The views consist of the tuples which are in R_1 *or* R_2. To derive the view constraints, we first consider the ambiguity of the insertion operation on the view. When a tuple is inserted into the view, the system will not know which base relation(s) the tuple will be translated to. Thus, the view constraint can not be derived. We introduce an extra attribute in the view, named *locality attribute*, to avoid the ambiguity. The domain of the attribute may contains $\{1, 2, 3\}$, with 1 and 2 representing that the corresponding tuples exist in R_1 and R_2, respectively, and 3 representing that the tuples exist in both. The view constraints can be derived by the following rules.

Property 12 Let a **union** view be defined as $\mathcal{V} = R_1 \cup R_2$. The constraints C_1 and C_2 on relation R_1 and R_2, respectively, can be represented as follows:

(C_1) $Q_{11}x_1 \in X_1 ... Q_{1i}x_i \in X_i ... Q_{1m}x_m \in X_m : \mathrm{S}_1 \Rightarrow \mathrm{T}_1$

(C_2) $Q_{21}y_1 \in Y_1 ... Q_{2j}y_j \in Y_j ... Q_{2n}y_n \in Y_n : \mathrm{S}_2 \Rightarrow \mathrm{T}_2$

Let $X_i = R_1$ and $Y_j = R_2$, without loss of generality. The view constraint can be derived by the rules shown in Table 10.

Conditions	Derivation Rules
	$\{\{\{Q_{11}x_1 \in X_1 ... Q_{1i}x_i \in \mathcal{V} ... Q_{1m}x_m \in X_m :$ $\mathrm{S}_1 \Rightarrow \mathrm{T}_1\} \wedge update(R_1)\}$ $\vee\{\{Q_{21}y_1 \in Y_1 ... Q_{2j}y_j \in \mathcal{V} ... Q_{2n}y_n \in Y_n :$ $\mathrm{S}_2 \Rightarrow \mathrm{T}_2\} \wedge update(R_2)\}\}$

Table 10: The constraint derivation rules for **union** view.

When there is an update (insertion/deletion/modification) on $\mathcal{V}$, the users must specify which relations to update in the locality attribute. If the value of the locality attribute is 1, $update(R_1) = true$ and $update(R_2) = false$. If it is 2, $update(R_1) = false$ and $update(R_2) = true$. Otherwise, it is 3, and $update(R_1) = true$ and $update(R_2) = true$.

3.7 Cartesian Product ($\times$)

The view constructed by the relational operation **Cartesian product** can be defined as $R_x \times R_y$. Updates on $\mathcal{V}$ have to follow the $cross-references$ condition [15]. The derivation rules are described in Table 11.

Property 13 Let the **Cartesian product** view be defined as $\mathcal{V} = R_x \times R_y$. Consider the constraint:

(C) $Q_1x_1 \in R_1\ Q_2x_2 \in R_2\ ...\ Q_mx_m \in R_m : \mathrm{S} \Rightarrow \mathrm{T}$

The view constraint can be derived by the rules shown in Table 11.

Conditions	Derivation Rules
$R_i = R_x$	$R_i \leftarrow \mathcal{V}, 1 \le i \le m$
$R_j = R_y$	$R_j \leftarrow \mathcal{V}, 1 \le i \le m$
otherwise	-

Table 11: The constraint derivation rules for **Cartesian product** view.

3.8 Outerjoin ($\overset{\circ}{\bowtie}$)

The view constructed by the relational operation **outerjoin** (it refers to *full-outer-nature-join*) [6] can be defined as $\mathcal{V} = R_x \overset{\circ}{\bowtie} R_y$, where $\mathcal{V}$ has the attribute set $K \cup C \cup D_x \cup D_y$ $(K \neq \emptyset)$, where $K \cup C \cup D_x$ is the attribute set of R_x, $K \cup C \cup D_y$ is the attribute set of R_y, K and C are the common attribute sets of R_x and R_y, and K is a *key* attribute set of R_x and R_y while C is not. To derive the view constraints, we divide the discussions into two parts as follows according to whether C is an empty set.

3.8.1 $C = \emptyset$

When C is an empty set, the join attributes contains only key attribute set. The two tuples of R_x and R_y with the same key attribute value will be merged into one tuple. R_x or R_y involved in a constraint will be replaced by $\mathcal{V}$. We obtain the derivation rules as shown in Table 12.

Property 14 Consider the **outerjoin** view, defined as $\mathcal{V} = R_x \overset{\circ}{\bowtie} R_y$, where the relations R_x and R_y have only the same key attribute set. Consider the constraint IC:

$$(IC) \quad Q_1 x_1 \in R_1 \; Q_2 x_2 \in R_2 \; \ldots \; Q_m x_m \in R_m : \mathrm{S} \Rightarrow \mathrm{T}$$

The view constraint can be derived by the rules shown in Table 12.

Conditions	Derivation Rules
$\{R_i = R_x\} \vee \{R_i = R_y\}$	$R_i \leftarrow \mathcal{V}$, $1 \le i \le m$
otherwise	–

Table 12: The constraint derivation rules for **outerjoin** views.

3.8.2 $C \neq \emptyset$

For any constraint IC involving R_x or R_y, if IC does not contain any attribute in the common attribute set C, the view constraint can be derived as the previous discussion (Section 3.8.1). Otherwise, the view constraint derivation rules are similar to the rules in **union** view.

4 Global Constraint Enforcement

When an update occurs, the system checks the constraints to see whether they are violated. To check the constraints, local data accesses may be needed depending on the types of the constraints. An update may be a deletion, an insertion, or a modification. A modification can be replaced as a deletion followed by an insertion. In this section, the checking of each category of assertions for insertion and deletion will be presented.

4.1 1V1R Assertions

Let the 1V1R view constraints on the view $\mathcal{V}$ be represented as

$$(C) \quad \forall x \in \mathcal{V} : \mathrm{S} \Rightarrow \mathrm{T}.$$

Consider the checking of the constraint C as shown as follows:

- **insert** a tuple t to $\mathcal{V}$

  step 1 check S(t) (S(t) is a boolean function. It returns true if tuple t satisfies the antecedent S of C; otherwise it returns false.): If it is true, do step 2; otherwise the tuple t trivially satisfies the constraint C.

  step 2 check T(t) (T(t) is similar to S(t)): If it is true, the tuple t nontrivially satisfies the constraint C and the insertion is accepted; otherwise the tuple does not satisfy the constraint C.

- **delete** a tuple from $\mathcal{V}$

  To delete a tuple from $\mathcal{V}$ need not check the 1V1R constraints.

4.2 2V1R Assertions

Consider a 2V1R constraint on $\mathcal{V}$:

$$(FD) \quad \forall x \in \mathcal{V} \, \forall y \in \mathcal{V} : x.A = y.A \Rightarrow x.B = y.B,$$

where A, B are the attributes of $\mathcal{V}$. Consider the checking of the FD as follows:

- **insert** a tuple t to $\mathcal{V}$

  step 1 perform a query against the corresponding base relation, say R, and store the result in Δ as:

  $$\Delta \leftarrow \Pi_B(\sigma_{A=t.A}(R))$$

  step 2 check Δ: If $\Delta = \emptyset$, t trivially satisfies FD; if $\Delta = t.B$, t nontrivially satisfies FD; otherwise, t does not satisfy FD.

- **delete** a tuple from $\mathcal{V}$

  To delete a tuple from $\mathcal{V}$ need not check the functional dependency constraint.

4.3 2V2R Assertions

Consider a 2V2R constraint as:

$$(RI) \quad \forall x \in \mathcal{V}_1 \; \exists y \in \mathcal{V}_2 : x.A = y.A$$

Consider the checking of RI as follows:

- **insert** a tuple t to $\mathcal{V}_1$

  step 1 perform a query against the corresponding base relation of $\mathcal{V}_2$, say R, and store the result in Δ as:

  $$\Delta \leftarrow \sigma_{A=t.A}(R)$$

  step 2 check $|\Delta|$: If $|\Delta| \neq 0$, t nontrivially satisfies RI; otherwise, t does not satisfy RI.

- **delete** a tuple t from $\mathcal{V}_2$

  step 1 perform a query against the corresponding base relation of $\mathcal{V}_1$ and $\mathcal{V}_2$, say S and R, respectively as:

  $$\Delta_1 \leftarrow \sigma_{A=t.A}(R) \text{ and } \Delta_2 \leftarrow \sigma_{A=t.A}(S)$$

  step 2 check $|\Delta_1|$ and $|\Delta_2|$: If $|\Delta_1| = 0$, the deletion satisfies RI; if $|\Delta_1| \neq 0$ and $|\Delta_2| > 1$ the deletion satisfies RI; otherwise, does not satisfy RI.

- **delete** a tuple from $\mathcal{V}_1$ and **insert** a tuple to $\mathcal{V}_2$

  To delete a tuple from $\mathcal{V}_1$ and insert a tuple to $\mathcal{V}_2$ need not check the referential integrity constraint.

5 Conclusion

A set of rules for the derivation of the global constraints in multidatabase systems from the individual local constraints are proposed. The constraints represented by assertions are classified into a number of categories. The global schema is a view. According to various views constructed from different relation operations, we derive the corresponding view constraints via the derivation rules for various categories of the constraints. We also consider the enforcement of the view constraints. To check the global constraints for an update, the systems may need to query the local databases. The checking of the 1V1R constraints does not need to access databases while the others do. In the future, we will consider more complicated constraints and more efficient strategies to enforce the constraints.

References

[1] P. A. Bernstein, B. T. Blaustein, and E. M. Clarke. Fast Maintenance of Semantic Integrity Assertions Using Redundant Aggregate Data. In *Proc. 6th VLDB conference*, pages 126–136, 1980.

[2] Y. Breitbart. Multidatabase Interoperability. *SIGMOD RECORD*, 19(3):53–60, 1990.

[3] C. S. Chang and A. L. P. Chen. Determining Probabilities for Probabilistic Partial Values. In *Proc. of the International Conf. on Data and Knowledge Systems for Manufacturing and Engineering (DKSME)*, pages 277–284, 1994.

[4] A. L. P. Chen. Optimizing Multidatabase Queries by Query Transformation. *Journal of Information Science and Engineering*, 7(2):279–296, 1991.

[5] A. L. P. Chen, D. Brill, M. Templeton, and C. Yu. Distributed Query Processing in a Multiple Database System. *IEEE Journal on Selected Areas in Communications*, 7(3):390–398, 1989.

[6] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley, 5th Ed., 1990.

[7] U. Dayal and Philip A. Bernstein. On the Correct Translation of Update Operations on Relational Views. *ACM Trans. on Database Systems*, 8(3):381–416, 1982.

[8] U. Dayal and H.Y. Hwang. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Tran. Software Eng.*, 10(6):628–644, 1984.

[9] P. Drew, R. King, and D. McLeod. Report of the Workshop on Semantic Heterogeneity and Interoperation in Multidatabase Systems. *SIGMOD RECORD*, 22(3):47–56, 1993.

[10] A. Gupta and J. Widom. Local Verification of Global Integrity Constraints in Distributed Databases. In *Proc. of the ACM SIGMOD*, pages 49–58, 1993.

[11] A. M. Keller. The Role of Semantics in Translating View Updates. *IEEE Computer*, 19:63–74, 1986.

[12] J. J. King. QUIST: A System for Semantic Query Optimization in Relational Databases. In *Proc. 7th VLDB conference*, pages 510–517, 1981.

[13] J.L. Koh and A.L.P. Chen. Integration of Heterogeneous Object Schemas. In *Proc. Entity-Relationship Approaches*, 1993.

[14] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of Multiple Autonomous Databases. *ACM Computing Surveys*, 22(3):237–293, 1990.

[15] Y. Masunaga. A Relational Database View Update Translation Mechanism. In *Proc. 10th VLDB conference*, pages 309–320, 1984.

[16] A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Tran. Software Eng.*, 13(7):785–798, 1987.

[17] X. Qian. Distribution Design of Integrity Constraints. *Expert Database Systems*, pages 75–84, 1988.

[18] M. P. Reddy, M. Siegel, and A. Gupta. Towards an Active Schema Integration Architecture for Heterogeneous Database Systems. In *Proc. Workshop on Interoperability in Multidatabase Systems*, pages 178–183, 1993.

[19] K. Tanaka and Y. Kambayashi. Logical Integration of Locally Independent Relational Databases into a Distributed Database. In *Proc. 7th VLDB conference*, pages 131–141, 1981.

[20] P. S. M. Tsai and A. L. P. Chen. Querying Uncertain Data in Heterogeneous Database. In *Proc. IEEE Int'l Workshop on Research Issues on Data Eng.*, 1993.

[21] F. S. C. Tseng, A. L. P. Chen, and W. P. Yang. Answering Heterogeneous Database Queries with Degrees of Uncertainty. *Distributed and Parallel Databases*, 1(3):281–302, 1993.

[22] C. T. Yu and W. Sun. Automatic Knowledge Acquisition and Maintenance for Semantic Query Optimization. *IEEE Trans. on Knowledge and Data Eng.*, 1(3):362–375, 1989.