Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases

Shailesh Agarwal[1], Arthur M. Keller[2], Gio Wiederhold[2], and Krishna Saraswat[3]

Abstract

In this work we address the problem of dealing with data inconsistencies while integrating data sets derived from multiple autonomous relational databases. The fundamental assumption in the classical relational model is that data is consistent and hence no support is provided for dealing with inconsistent data. Due to this limitation of the classical relational model, the semantics for detecting, representing, and manipulating inconsistent data have to be explicitly encoded in the applications by the application developer.

In this paper, we propose the *flexible relational* model, which extends the classical relational model by providing support for inconsistent data. We present a flexible relation algebra, which provides semantics for database operations in the presence of potentially inconsistent data. Finally, we discuss issues raised for query optimization when the data may be inconsistent.

1. Introduction

Advances in computer networking technology and the availability of economical computing hardware have led to a proliferation of autonomous databases connected by high speed communication networks. As a result of this greatly increased access to remote databases, a growing number of database applications need to jointly manipulate data located in multidatabases [Litwin89, Litwin90, Breitbart90, Sheth90, Bright92, Scheuermann94]. Since the component databases of a particular multidatabase are most likely autonomous, they tend to be heterogeneous with respect to each other. Further, the distribution of data among such databases is likely to be arbitrary, often redundant, and possibly inconsistent. Hence, the development and maintenance of applications that manipulate data from multiple databases is generally expensive and difficult. These applications have to explicitly resolve any heterogeneities, especially inconsistencies, among the data sets derived from these databases.

This paper focuses on the problems of manipulating data from multiple autonomous databases that may be mutually inconsistent. For the purposes of this work it is assumed that all other types of heterogeneities such as hardware, OS, network, or SQL language variations have been resolved via a homogenizing veneer on each individual database and also that each database presents a relational interface.

2. Definition Of Consistency

The term *inconsistency* has been used in the literature for several specific cases. One form of inconsistency occurs when two tuples match on all the key attribute values but have conflicting values for some non-key attributes [DeMichiel89]. Thus, for example, if the birth date of an employee differs in two databases then there is an inconsistency.

While the above notion of conflicts between tuples with matching key attribute values implies an inconsistency, the

lack of conflict between such tuples does not guarantee that the data is consistent [Ramarao88]. An approach for addressing such problems is discussed in [Ceri92].

Another form of inconsistency occurs when there is an error in the values of the primary key attributes themselves. The detection of such inconsistencies is, by itself, a difficult problem. A probabilistic reasoning approach for detecting such inconsistencies using data associated with non-key attributes is presented in [Chatterjee91].

In this paper we consider the first type of inconsistency, i.e., where tuples with matching values for primary key attributes conflict in their non-key attribute values. This notion of conflicting tuples is formalized in Definition 1.

Definition 1 Two tuples t_f and t_g associated with a relational schema (K,Z), where K is the entity identifying attribute set and Z is the non entity-identifying attribute set, are non-conflicting, denoted by $t_f \cong t_g$, if at least one of the following holds

- $t_f[K] \neq t_g[K]$.
- $t_f[K] = t_g[K]$ and $(\neg \exists\ C)\ (C \in Z \wedge t_f[C] \neq null \wedge t_g[C] \neq null \wedge\ t_f[C] \neq t_g[C])$.

Note that this form of inconsistency does not imply just a difference in format or scale of the data values but rather that tuples representing the same object have conflicting data values for the corresponding attributes.

3. Motivation

Relational databases are expected to be consistent. The consistency of each database is enforced by storing data in a normalized and non-redundant manner [Date83]. Several techniques have been developed for maintaining consistency among replicated data in distributed databases [Bernstein81, Gray81, Traiger82] and this is still an active area of research [Wiederhold90, Barbara91]. The point is that maintenance of consistency in centralized and also distributed relational databases is considered crucial in order to be able to manipulate data using the relational paradigm.

This paper is based on the premise that while, for tightly managed applications, such as airline reservation or automated teller systems in the banking industry, it is necessary and feasible (although non-trivial) to maintain consistency among tightly coupled databases, it is not feasible to do so among loosely coupled databases. For example, consider the case of integrating data from multiple autonomous databases that store related data in a manufacturing facility. These databases operate independently for long periods of time and easily become inconsistent with respect to each other. It is impractical to enforce consistency among such databases and hence additional semantics are required for jointly manipulating any inconsistent data derived from such databases.

Consider the example in Figure 1, where data from the two relations *R1* and *R2* is to be integrated. Each relation represents data about the same object set, i.e., *wafers* in a semiconductor manufacturing facility.

Attribute *Waferid* is the primary key attribute for each of the relations. The two tuples with *Waferid = 200* are inconsistent. In relation *R1* the value for attribute *Wtype* of

---

	R1				R2		

Waferid	Wtype	Dia
100	"N"	6.0
200	"P"	6.0
300	"P"	5.0

Waferid	Wtype	Dia
200	"N"	6.0
400	"N"	6.0
500	"P"	5.0

Figure 1: Relations to be integrated.

the tuple with _Waferid = 200_ is _"P"_ whereas in relation _R2_ the corresponding value is _"N"_. Suppose that an integrated view _R_ with schema (_Waferid, Wtype, Dia_) is to be defined over these two relations.

A possible specification for this integrated view is to take the union of these two source relations, i.e., $R = R1 \cup R2$. The materialized view _R_ is shown in Figure 2.

_R_

Waferid	Wtype	Dia
100	"N"	6.0
200	"P"	6.0
200	"N"	6.0
300	"P"	5.0
400	"N"	6.0
500	"P"	5.0

Figure 2: Materialized view _R_.

There are two tuples with the same primary key attribute value, i.e., _Waferid = 200_. This is not resolved by elimination of duplicates since these tuples are not identical. The set of tuples in this materialized view does not conform to the semantics of a classical relation.

Further, consider the problem of manipulating the data associated with the view _R_. Let us pose query _Q_ as follows:

select _Waferid, Wtype, Dia_
from _R_
where _Wtype = "P"_

Classical relational semantics returns the following tuples.

Waferid	Wtype	Dia
200	"P"	6.0
300	"P"	5.0
500	"P"	5.0

Figure 3: Result of query _Q_.

The classical relational query processing system does not have the semantics to recognize the fact that data associated with the tuple with _Waferid = 200_ is inconsistent. In fact, the tuple ⟨_200, "P", 6.0_⟩ may not even belong to the result if the actual value of its attribute _Wtype_ is _"N"_ and not _"P"_. Thus, there is a loss of information and a potential for error since the user is not informed of this inconsistency.

An option for the view definer is to try to detect and resolve all inconsistencies before the integration. The problem is that due to the large amounts of data stored within typical databases, detection of all the database inconsistencies is an impossibly expensive proposition. Furthermore, even if all the inconsistencies were detected and resolved at a given time, future independent updates can still introduce new inconsistencies. Thus, dealing with such inconsistencies within the classical relational framework is an open problem.

This paper presents another option. It proposes the notion of _flexible relation_ for addressing some of these problems of inconsistent data. This notion is formalized by defining the flexible relational model, which extends the classical relational model by supporting the representation of inconsistent data and also providing extended semantics for data manipulation operators in the presence of such inconsistent data. The basic motivation for the flexible relational model is to provide a set of semantics that enable the representation and manipulation of possibly inconsistent sets of data retrieved from multiple sources. An important objective of these semantics for inconsistent data is to maximize the information presented to the users. This paper also presents and discusses some new issues raised for processing and optimizing queries in an environment where data from multiple sources can be potentially inconsistent.

4. Flexible Relational Model

The basic data structure in the flexible relational model is a _flexible relation_. This section focuses on the definition of the flexible relation while the next section considers semantics of database operations over such flexible relations.

A flexible relation is a derived relation. The schema and data of a flexible relation are derived from a classical relation that is in Boyce-Codd normal form. A flexible relation is derived from a classical relation by applying an operator _flexify_ denoted by f.

4.1 Flexify

Operator _flexify_ (f) takes a classical relation as input and converts it to a flexible relation. The schema of the resulting flexible relation is derived by extending the schema of the input classical relation with three ancilliary attributes, _Cons, Sel,_ and _Src,_ which are described in Section 4.3. Each tuple of the classical relation is converted to a tuple of the flexible relation. Appropriate values are provided for the ancilliary attributes of these flexible relation tuples.

Definition 2 A flexible relation _FR_ obtained by applying the _flexify_ (f) operator to a classical relation _R_ with schema (_K,Z_), where _K_ is the entity-identifying attribute set, _Z_ is the non entity-identifying attribute set, and _R_ is associated with a set of tuples $\{t_i\}$, is denoted by $FR = f(R)$, and is such that it has a schema (_K,Z,Cons,Sel,Src_), where _Cons_ is the consistency status attribute, _Sel_ is the selection status attribute, _Src_ is the source attribute and it is associated with the following set of tuples

$\{t'_i \mid t'_i[K] = t_i[K], \forall A \in Z, t'_i[A] = t_i[A], t'_i[Cons] = cons, t'_i[Sel] = true, t'_i[Src] = "R"\}$

Thus, as shown in Figure 4, a flexible relation is derived

_R_

K	Z1	Z2	Z3
10	x	null	z
20	y	v	z
30	null	w	z

$FR = f(R)$

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	null	z	cons	true	R
20	y	v	z	cons	true	R
30	null	w	z	cons	true	R

Figure 4: Derivation of a flexible relation.

from a classical relation by extending its schema with the ancilliary attributes and assigning values for these ancilliary attributes for each of the tuples. (See Section 4.3 for a more detailed description of these ancilliary attributes.)

A classical relation by definition is consistent and hence a flexible relation derived from a single classical relation is also consistent. However, when data from one or more individually consistent flexible relations are merged, inconsistencies may arise. The notion of a *cluple* is introduced in order to represent inconsistent data in a flexible relation.

4.2 Cluples

A *cluple* is defined as a <u>cluster of tuples</u> such that all the tuples in that cluster match on their original entity-identifying attribute values, which were used to define some object uniquely.

Definition 3 A cluple c with schema ($K,Z,Cons,Sel,Src$) is a cluster of tuples with the same schema and where all tuples match on all entity identifying attribute values, i.e., for any two tuples t_p and t_q from the cluple, $t_p[K] = t_q[K]$.

A flexible relation is defined as *a set of cluples* where each cluple represents data about a particular entity. Note that within a cluple each tuple has the same set of values for the attribute set K. Hence, in the context of a cluple the primary key for the tuples is the concatenation of the attribute set K and the ancilliary attribute Src. Each tuple in a given cluple has a unique value for the attribute Src and this value refers to the original source relation from which that tuple was derived. Depending on the tuples associated with a cluple, the cluple may be either *consistent* or *inconsistent*.

A definition for the consistency of tuples in the context of classical relations was presented earlier (Definition 1). That definition is now restated in the context of flexible relations.

Definition 4 Two tuples t_f and t_g associated with a cluple with schema ($K,Z,Cons,Sel,Src$), where K is the entity-identifying attribute set, Z is the non entity-identifying attribute set, $Cons, Sel,$ and Src are the ancilliary attributes, are non-conflicting, denoted by $t_f \cong t_g$, if the following holds

$(\neg \exists\ C)\ (C \in Z \wedge t_f[C] \neq null \wedge t_g[C] \neq null \wedge\ t_f[C] \neq t_g[C])$.

Note that in the above definition, the ancilliary attributes are not considered while determining the consistency between tuples. In fact, the value for the ancilliary attribute $Cons$ is determined by evaluating the consistency of the tuples associated with a given cluple.

Definition 5 A cluple c is *consistent* if it contains only non-conflicting pairs of tuples. Formally, a cluple c is consistent if all pairs of tuples in the cluple are non-conflicting, i.e., $(\forall t_f)\ (\forall t_g)\ (t_f \in c \wedge t_g \in c \wedge t_f \cong t_g)$. A cluple containing only one tuple is, by definition, consistent.

Equivalently, a cluple is consistent if for each attribute, all of the non-null values agree.

	K	Z1	Z2	Z3
t1	10	x	null	z
t2	10	x	y	z
t3	10	x	w	z

	K	Z1	Z2	Z3
t4	20	y	null	z
t5	20	y	y	null

Figure 5: Examples of cluples.

Consider the cluples c and c' in Figure 5, with schema ($K,Z1,Z2,Z3$), ignoring the ancilliary attributes, where K is the entity-identifying attribute and $Z1, Z2,$ and $Z3$ are the non entity-identifying attributes.

Cluple c is inconsistent since it contains at least one conflicting tuple pair, i.e., ($t2, t3$). On the other hand, cluple c' is consistent since it has only non-conflicting tuple pairs. Thus, the notion of a cluple enables the representation of inconsistent data.

While merging data from multiple sources one has to deal with the problem of incomplete or missing information. A widely used approach is to use the value *null* [Codd79] to represent missing information. While there are many different interpretations of null values [ANSI75], we have chosen the value *null* to have an interpretation of *no information* [Zaniolo84], where such a null can be a place holder for either a nonexistent or an unknown value.

4.3 Ancilliary attributes

As stated earlier, in addition to the user defined attributes, the schema for a flexible relation also includes three ancilliary attributes: (i) *Cons*, (ii) *Sel*, and (iii) *Src*. These ancilliary attributes are instantiated by the application of the *flexify* operator. Each tuple of a flexible relation has a value for each of these attributes and these values are managed by the system.

Consider the set of source relations in Figure 6 and a materialized flexible relation in Figure 7, which is derived from these source relations.

S1

K	Z1	Z2	Z3
10	x	null	z
20	y	x	v

S2

K	Z1	Z2	Z3
10	x	y	z

S3

K	Z1	Z2	Z3
10	x	w	z
20	y	null	v

Figure 6: Source relations.

$FR = f(S1) \cup f(S2) \cup f(S3)$

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	null	z	incons	true	S1
10	x	y	z	incons	true	S2
10	x	w	z	incons	true	S3
20	y	x	v	cons	true	S1
20	y	null	v	cons	true	S3

Figure 7: Example of a materialized flexible relation.

The interpretation and use of data associated with these three ancilliary attributes is discussed below.

Cons: Attribute *Cons* refers to the consistency status of the cluple to which the tuple belongs. Hence, all tuples associated with a particular cluple have the same value for attribute *Cons*. The domain of this attribute has two values {*cons, incons*}.

For example, in Figure 7, the cluple with $K = 10$ is inconsistent, while the cluple with $K = 20$ is consistent.

Sel: Attribute *Sel* denotes the selection status of the cluples present in the result. The value of this attribute is determined by applying the selection predicate to the data in each of the cluples. Similar to the previous case of consistency status, all tuples of a cluple have the same selection status value. The domain of attribute *Sel* consists of three values {*true*, *maybe*, *false*}. However, since cluples with a value of *false* for attribute *Sel* are rejected, the results returned for a query never have cluples with *Sel = false*. The process of evaluating selection predicates over cluples is discussed in Section 5.1.3.

For the example in Figure 7, since the selection did not have any restriction clause, all the cluples have a selection status of *true*.

Src: Attribute *Src* refers to the source relation from which a particular tuple is derived. This is similar to the notion of attribute *locality* as defined in [Gamal-Eldin88]. In Figure 7, values *S1, S2,* and *S3* refer to the source relations from which the tuples are derived.

In case of inconsistent data, the source information is useful for determining the cause of a particular inconsistency and subsequently its resolution.

Thus, these three ancillary attributes provide information that is useful to the user for detecting inconsistencies (*Cons*), for interpreting the results with respect to the selection status (*Sel*), and also for resolving these inconsistencies using the source information (*Src*).

5. Flexible Relational Algebra

In the previous section, the notion of a flexible relation as a set of cluples was presented. In this section, the semantics of databases operations over such flexible relations are presented.

One of the factors complicating the semantics of these operations is the possibility of inconsistent data in the form of inconsistent cluples. The intent of these semantics is to perform meaningful operations in the presence of inconsistent data and also to provide as much information as possible to enable the user to resolve this inconsistency.

The full algebra for flexible relations is defined in [Agarwal92]. Due to space limitations, in this paper we describe only some of the operations in this algebra. Operations over flexible relations eventually translate to operations over cluples. Hence, semantics for operations relevant to cluple processing are defined first in Section 5.1 and then these semantics are extended for operations over flexible relations in Section 5.2.

5.1 Cluple Operations

The set of cluple operations defined in [Agarwal92] includes *merging, equivalence, selection, union, cartesian product,* and *projection*. The following subsections consider all of these cluple operations except *cartesian product* and *projection*.

5.1.1 Merging

The merge operator merges the tuples in a cluple to a single nested tuple, which is referred to as a *merged cluple*.

Definition 6 A merged cluple obtained by applying the merge operator Ξ to a cluple $c = \{t_1, t_2, ..., t_n\}$ with schema $(K, Z, Cons, Sel, Src)$, is denoted by $\Xi(c)$, and is such that

$\Xi(c)[K] = t_1[K] = t_2[K] = \cdots = t_n[K], \forall A \in Z, \Xi(c)[A] = t_1[A] \cup t_2[A] \cup \cdots \cup t_n[A], \Xi(c)[Cons] = t_1[Cons] = t_2[Cons] = \cdots = t_n[Cons], \Xi(c)[Sel] = t_1[Sel] = t_2[Sel] = \cdots = t_n[Sel]$, and $\Xi(c)[Src] = t_1[Src] \cup t_2[Src] \cup \cdots \cup t_n[Src]$.

The merged cluple $\Xi(c)$ is further refined by subsuming the null values associated with a particular attribute of $\Xi(c)$ by any non-null value associated with that attribute [Zaniolo84]. Thus, $\Xi(c)[A] = null$ if and only if $\forall i \; t_i[A] = null$.

Consider the cluple c in Figure 8, where K is the entity-identifying attribute and *Z1, Z2,* and *Z3* are the non entity-identifying attributes.

$$c$$

K	$Z1$	$Z2$	$Z3$	$Cons$	Sel	Src
10	*x*	*y*	*null*	*incons*	*true*	*S1*
10	*u*	*null*	*null*	*incons*	*true*	*S2*
10	*u*	*v*	*null*	*incons*	*true*	*S3*

Figure 8: Example cluple *c*.

The merged cluple obtained by merging cluple c is $\Xi[c] = \langle 10, \{x, u\}, \{y, v\}, \{null\}, incons, true, \{S1, S2, S3\} \rangle$. For attribute *Z2*, the null value is subsumed by the non-null values but for attribute *Z3* there is no non-null value and so it is associated with a null value.

As discussed in the next section the merging operation is useful for determining the equivalence of two cluples.

5.1.2 Equivalence

Cluples instantiated from different sets of sources or different selection conditions applied over the same set of sources may match in their key attribute values. A notion of equivalence is defined for such cluples based on the values of their non-entity identifying and the ancillary attributes. Two cluples associated with a particular schema are defined to be equivalent if the merged cluples derived by merging each of the cluples are equivalent. This form of equivalence is weaker than requiring each cluple to have the same set of tuples.

Definition 7 Two merged cluples $\Xi(c_p)$, and $\Xi(c_q)$ associated with a schema $(K, Z, Cons, Sel, Src)$ are considered equivalent, denoted by $\Xi(c_p) \equiv \Xi(c_q)$ if $\Xi(c_p)[K] = \Xi(c_q)[K]$ and $\forall A \in Z$, either $\Xi(c_p)[A] = \Xi(c_q)[A]$ or $(\Xi(c_p)[A] = null \wedge \Xi(c_q)[A] = null)$, and $\Xi(c_p)[Cons] = \Xi(c_q)[Cons]$ and $\Xi(c_p)[Sel] = \Xi(c_q)[Sel]$. In cases where attribute A is associated with more than one value, the term $\Xi(c_p)[A] = \Xi(c_q)[A]$ implies that the set of values associated with $\Xi(c_p)[A]$ is the same as the set of values associated with $\Xi(c_q)[A]$.

Note that while comparing such merged cluples the ancillary attribute *Src* is not considered. Thus, two merged cluples are considered equivalent if they match exactly in each of their attributes other than the ancillary attribute *Src*.

Definition 8 Two cluples c_p and c_q with schema $(K, Z, Cons, Sel, Src)$, are considered equivalent, denoted by $c_p \equiv c_q$, if the merged form of these cluples are equivalent, i.e., $\Xi(c_p) \equiv \Xi(c_q)$.

5.1.3 Selection

In classical relational algebra the select operator determines the selection status of the tuple for a given selection condition. This condition is specified by means of a selection predicate.

Definition 9 Given a classical relational schema (K,Z), a *simple predicate* is of the form $(A \; op \; \lambda)$ or $(A1 \; op \; A2)$, where $A, A1, A2 \in K \cup Z$, $op \in \{=, \neq, >, \geq, <, \leq\}$ and λ is a single value, i.e., $\lambda \in DOM(A) \cup null$, where $DOM(A)$ refers to the domain of attribute A.

Definition 10 A selection predicate q is in conjunctive normal form (CNF) if it has the form $p_1 \wedge p_2 \wedge ... \wedge p_i$ where each p_i is a predicate such that no p_i contains $\wedge$ and negation applies only to individual comparisons [Maier83].

For classical relations, the selection predicate over a tuple evaluates to a boolean answer of either *true* or *false* for that tuple. In order to apply the selection predicate to a cluple c, the selection predicate is applied to the nested tuple $\Xi[c]$, which is obtained by merging cluple c. Semantics of the selection predicate as defined above (Definition 9) are only adequate for selecting 1NF tuples, i.e., where each attribute is associated with exactly one value.

However, upon merging an inconsistent cluple, a nested tuple is obtained. So an extended set of semantics for specifying and evaluating selection predicates over such nested tuples is required.

Consider the example of cluple $c1$ in Figure 9.

$c1$

	K	Z1	Z2	Z3	Cons	Sel	Src
t1	10	x	null	y	incons	true	S1
t2	10	x	y	y	incons	true	S2
t3	10	x	w	y	incons	true	S3

Figure 9: Example cluple $c1$.

The merged form of cluple $c1$ is $\Xi[c1] = \langle 10, \{x\}, \{y,w\}, \{y\}, incons, true, \{S1,S2,S3\} \rangle$. In order to evaluate selection predicates over such a nested tuple the definition of the selection predicate is extended to allow conditions over attributes that may be associated with more than one value. An example of a selection predicate for cluple $c1$ is $[(Z1 = x) \wedge (Z2 = Z3)]$. The notion of a simple partial predicate is introduced to express these semantics.

Definition 11 Given a flexible relational schema $(K,Z,Cons,Sel,Src)$, a *simple partial predicate* is of the form $(A \; op \; \lambda)$ or $(A1 \; op \; A2)$, where $A, A1, A2 \in K \cup Z$, A is associated with a set of values $\{\delta_i\}$, $A1$ is associated with a set of values $\{\delta 1_j\}$ and $A2$ is associated with a set of values $\{\delta 2_k\}$, $op \in \{=, \neq, >, \geq, <, \leq\}$ and λ is a single value, i.e., $\lambda \in DOM(A) \cup null$, and for each of these sets $\{\delta_i\}$, $\{\delta 1_j\}$ and $\{\delta 2_k\}$, the null values has been subsumed by any non-null value in that set.

The predicate $(A \; op \; \lambda)$ evaluates to either *true*, *false*, or *maybe* as follows:

- *true*, if $(\forall i)$, $(\delta_i \; op \; \lambda)$ is *true*.

- *false*, if $(\neg\exists i)$, such that $(\delta_i \; op \; \lambda)$ is *true*.
- *maybe*, if $(\exists i)$, such that $(\delta_i \; op \; \lambda)$ is *true* and $(\exists i)$, such that $(\delta_i \; op \; \lambda)$ is *false*.

The predicate $(A1 \; op \; A2)$ evaluates to either *true*, *false* or *maybe* as follows:

- *true*, if $(\forall j) \; (\forall k)$, $(\delta 1_j \; op \; \delta 2_k)$ is *true*.
- *false*, if $(\neg\exists j) \; (\neg\exists k)$, such that $(\delta 1_j \; op \; \delta 2_k)$ is *true*.
- *maybe*, if $(\exists j) \; (\exists k)$, such that $(\delta 1_j \; op \; \delta 2_k)$ is *true* and $(\exists j) \; (\exists k)$, such that $(\delta 1_j \; op \; \delta 2_k)$ is *false*.

Following the semantics of no information nulls [Zaniolo84], any comparison with null values evaluates to false. Hence, predicate $(A \; op \; \lambda)$ evaluates to false if either attribute A is *null* or λ is *null*, and predicate $(A1 \; op \; A2)$ evaluates to *false* if either of the attributes $A1$ or $A2$ is *null*.

A selection predicate is a partial selection predicate if one of its simple predicates is a simple partial predicate. The selection predicate $[(Z1 = x) \wedge (Z2 = Z3)]$ for cluple $c1$ is an example of a partial selection predicate since attribute $Z2$ is associated with more than one value, i.e., $\{y, w\}$.

The status of a partial selection predicate is determined by the status of all of its predicates. The logic for evaluating the status of a predicate consisting of partial predicates is defined by the truth tables in Figure 10.

α	not α
t	f
m	m
f	t

α ∨ β		β		
		t	m	f
	t	t	t	t
α	m	t	m	m
	f	t	m	f

α ∧ β		β		
		t	m	f
	t	t	m	f
α	m	m	m	f
	f	f	f	f

Figure 10: Truth tables for three-valued logic.

In these tables, *t* stands for *true*, *f* for *false*, and *m* for *maybe*. Symbols α and β refer to predicates.

This definition for a partial predicate and its evaluation is influenced by similar notions for selecting over attributes that may be associated with sets of values. The three-valued logic as defined below and the truth tables for three-valued logic (Figure 10) are identical to the results obtained previously in the context of *nulls* under the "unknown" interpretation [Codd79] and *partial values* [DeMichiel89]. The notion of a partial value as a finite set of values is just a special case of a null value that is used to represent the entire domain for a particular attribute. Our notion of a *maybe* selection and the three-valued logic as shown in Figure 10, while influenced by these efforts, is semantically quite different from them.

In flexible relations, attributes are associated with multiple values due to data conflicts. Such conflicts, as defined in our

work, result from violations of functional dependencies. For example, two tuples $(\langle 10,"z"\rangle, \langle 10,"x"\rangle)$ corresponding to schema (K,Z), where there is a dependency of the form $K \rightarrow Z$, are inconsistent. The merged cluple represents the information in these two tuples as $(\langle 10, ["z","x"]\rangle)$. It is tempting to interpret the set of values $["z","x"]$ either as a set null or a partial value. However, it is important to note that the set of values in our example does not conform to any of these interpretations. A set null or a partial value for an attribute corresponds to a set of values such that the true value for that attribute is exactly one of the values in that set. Thus, a set null or partial value is used to represent an *under-constrained* system, i.e., a finite set of values is available but it is not known which of the values is the real value. On the other hand, the set of values $["z","x"]$ represents the fact that each of the values is known to be true by their respective sources and so there is a conflict in the merged data. Hence, this set represents an *over-constrained* system, i.e., different values are believed to be true for a particular attribute of a tuple. Thus, while the work related to set nulls and partial values is not directly applicable to our work, still it is important to understand the distinction between them.

The value of the status attribute, *Sel*, of a cluple is the selection status value of the overall predicate applied to that cluple. The result of applying a new selection predicate q to a particular cluple c, is denoted by $\gamma_q(c)$ which is combined with the existing value of attribute *Sel* of that cluple using the logic defined below. Note that the value of *Sel* may change as a result of the selection operation. Thus the selection operation over a cluple c associated with schema (K,Z), denoted by $\sigma_q(c)$, where q is the selection predicate, results in a new cluple c' as follows:

$$\{c' \mid (c'[K] = c[K]) \wedge (c'[Z] = c[Z])$$
$$\wedge (c'[Cons] = c[Cons])$$
$$\wedge (c'[Sel] = (\gamma_q(c) \wedge c[Sel]))$$
$$\wedge (c'[Src] = c[Src]) \}.$$

As described above, a selection predicate over a cluple can evaluate to *true, false,* or *maybe*. Let us try to understand the meaning of a *maybe* selection. A *maybe* selection for a cluple implies that while the selection predicate evaluates to *true* based on some data in that cluple, there is also some other data in the same cluple such that the selection predicate evaluates to *false*. This situation arises since there is an inconsistency in the cluple and the selection predicate is over one or more attributes that are associated with conflicting data.

For the cluple $c1$ in Figure 8, the selection predicate $[(Z1 = x) \wedge (Z2 = Z3)]$ evaluates to *maybe* since one of the attributes in the predicate, i.e., $Z2$, is associated with conflicting data.

5.1.4 Union

The union operation combines the tuples of the two source cluples to form a new cluple. This operation is meaningful only when the source cluples represent data about the same entity and are in union compatible form.

In flexible relational algebra, the union operator has to be applied before applying any selection operator to the source cluples. Consider a cluple $c1$ selected after the application of the selection predicate $(Z1 = x)$.

c1

K1	Z1	Z2	Cons	Sel	Src
10	x	y	cons	true	S1
10	x	null	cons	true	S2

Figure 11: Selected cluple *c1*.

Consider another cluple *c2* selected after the application of the selection predicate $(Z2 = v)$.

c2

K1	Z1	Z2	Cons	Sel	Src
10	z	v	cons	true	S3

Figure 12: Selected cluple *c2*.

Taking the union of *c1* and *c2* and then evaluating the predicate $[(Z1 = x) \vee (Z2 = v)]$ results in the cluple shown in Figure 13.

c1 $\cup$ c2

K1	Z1	Z2	Cons	Sel	Src
10	x	y	incons	maybe	S1
10	x	null	incons	maybe	S2
10	z	v	incons	maybe	S3

Figure 13: Selected cluple *c1 $\cup$ c2*.

Even though both the source cluples have a selection status of *true*, the result has a selection status of *maybe*.

Another, more severe problem, is that the application of the selection operator before the union operation may result in the elimination of cluples that logically belong to the final result, i.e., some cluples that are eliminated from the source relations may actually be required to be part of the answer. Such a situation arises because of the possibility of inconsistencies between the source cluples. As discussed later (Section 6), this obviously has significant implications with respect to the strategies for optimizing flexible relation query expressions.

The union operation is valid only for those cases where the selection operator has not been applied over the source cluples. Note that in such cases, attribute *Sel* has a value of *true* by definition (Definition 2). The semantics of a union operator are formalized in the following definition.

Definition 12 A union of two cluples *c1* and *c2* associated with a schema $(K,Z,Cons,Sel,Src)$ where $c1[K] = c2[K]$, denoted by $c = c1 \cup c2$, is such that for each tuple $t \in c$ either $t \in c1$ or $t \in c2$.

The consistency of the resulting cluple, denoted by ancillary attribute *Cons*, has to be evaluated after the union operation. Even if each of the source cluples is independently consistent, it is still possible for the union of these cluples to be inconsistent. Ancilliary attribute *Sel* has the value *true* since it is assumed that no selection predicate has been applied to the cluple. Also, each tuple in the result cluple retains its source value, i.e., value of ancilliary attribute *Src*.

Consider the two cluples *c1* and *c2* in Figure 14.

c1

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	y	z	cons	true	S1
10	x	null	z	cons	true	S2

c2

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	null	z	cons	true	S3
10	x	h	null	cons	true	S4

Figure 14: Example cluples c1, c2.

The union of these cluples $c = c1 \cup c2$ is shown in Figure 15.

c1 $\cup$ c2

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	y	z	incons	true	S1
10	x	null	z	incons	true	S2
10	x	null	z	incons	true	S3
10	x	h	null	incons	true	S4

Figure 15: Cluple c1 $\cup$ c2.

Even though, each cluple c1 and c2 is consistent by itself, the union of these cluples $c = c1 \cup c2$ is inconsistent.

5.2 Flexible relation operations

As defined earlier, a materialized flexible relation is a set of cluples. If each cluple of a flexible relation is consistent then the flexible relation reduces to a classical relation. However, if a flexible relation is associated with one or more inconsistent cluples then extended semantics are required to operate over such relations.

In the previous section, the semantics for operations over cluples as operands were defined. This section defines the semantics for operations over flexible relations. These semantics are fully compatible with the classical relational algebra semantics.

In order to differentiate the extended operator symbols, each classical relational operator symbol is marked by ' ° ' to denote a *flexible relational* operator. Thus, for example, the symbol $\cup^\circ$ denotes the flexible union operator.

5.2.1 Membership

A notion of membership is defined for determining whether a given cluple *c* is a member of a particular flexible relation *FR*.

Definition 13 A cluple *c* is a member of a flexible relation *FR* if and only if there exists a cluple *c'* in flexible relation *FR* such that *c* is equivalent to *c'*.

This definition is formalized as follows:

$$\{c \in FR \Rightarrow (\exists c')(c' \in FR \land c \equiv c')\}.$$

5.2.2 Union

Consider two source flexible relations $FR_f(K,Z)$ and $FR_g(K,Z)$, where K is the set of entity-identifying attributes and Z is the set of non entity-identifying attributes. Each flexible relation may contain both types of cluples, i.e., consistent and inconsistent.

The result of the union operation $FR_f \cup^\circ FR_g$ is

$\{c \mid ((c \in FR_f \land (!\exists c_g) (c_g \in FR_g \land (c[K] = c_g[K])))$

$\lor (c \in FR_g \land (!\exists c_f) (c_f \in FR_f \land (c[K] = c_f[K])))$

$\lor (\exists c_f)(\exists c_g) (c_f \in FR_f \land c_g \in FR_g \land$

$(c[K] = c_f[K] = c_g[K]) \land (c = c_f \cup c_g)))\}.$

If the union of two mutually consistent classical relations has tuples that match in their key attributes, then they can only be duplicates. In this case, each such set of duplicate tuples is replaced by a single tuple. However, in case of flexible relations, tuples or cluples derived from different source relations may not be duplicates and in fact they can be inconsistent, hence the semantics is to combine these cluples to form a new cluple.

5.2.3 Selection

Cluples, whether consistent or inconsistent, that have a selection status of *false* are eliminated from the result. Thus, the result of selection over a flexible relation can be divided into two sets of cluples denoted as *true* or *maybe* respectively.

The *true* result of the selection operation $\sigma^\circ_q(FR)$, where q is the selection predicate, is

$$\{c \mid c \in FR \land (\sigma^\circ_q(c) = true)\}.$$

The *maybe* result of the selection operation $\sigma^\circ_q(FR)$, where q is the selection predicate, is

$$\{c \mid c \in FR \land (\sigma^\circ_q(c) = maybe)\}.$$

6. Flexible Relation Query Optimization

In this section the optimization of flexible relational queries is considered. The possibility of the presence of inconsistent data in a materialized flexible relation raises new issues for optimizing flexible relation queries. In this paper, we focus on the rules for pushing a selection over an union of flexible relations.

6.1 Pushing selection over union of flexible relations

$$(\sigma^\circ_q(FR1 \cup^\circ FR2)) \equiv (\sigma^\circ_q(FR1) \cup^\circ \sigma^\circ_q(FR2))$$

In order to investigate the validity of this rule, three cases, which cover all possibilities, are considered.

Case I: *FR1* and *FR2* are disjoint

If the flexible relations *FR1* and *FR2* are disjoint then no cluple from *FR1* can be merged with any cluple from *FR2*. So, the union of the two source relations does not result in the formation of any new cluples. Hence, the selection can be pushed to each source flexible relation, without losing any cluples in part or whole, and so the above rule is valid in this case.

Case II: *FR1* and *FR2* are not disjoint but consistent.

In this case, there may be cluples from *FR1* that match cluples in *FR2* in the key attribute values. Such cluples are individually consistent and also consistent with each other. Hence no new inconsistent cluples are created upon taking a union of the relations *FR1* and *FR2*.

Consider the example of flexible relations *FR1* and *FR2*, in Figure 16, which contain overlapping but consistent data.

FR1

K	Z1	Z2	Z3
10	x	null	null
10	null	y	null
20	x	y	z
30	x	null	z

FR2

K	Z1	Z2	Z3
10	x	null	z
20	x	null	z
30	x	p	null

Figure 16: Consistent flexible relations.

Further, consider a query $\sigma^\circ{}_q(FR1 \cup^\circ FR2)$ where $q = [Z2 = y \wedge Z3 = z]$ is the selection predicate. The result obtained without pushing the selection is shown in Figure 17.

$$\sigma_q^\circ(FR1 \cup^\circ FR2)$$

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	null	null	cons	true	FR1
10	null	y	null	cons	true	FR1
10	x	null	z	cons	true	FR2
20	x	y	z	cons	true	FR1
20	x	null	z	cons	true	FR2

Figure 17: Result without pushing selections to consistent flexible relations.

The result with pushing selections to the source relations is shown in Figure 18.

$$\sigma_q^\circ(FR1) \cup^\circ \sigma_q^\circ(FR2)$$

K	Z1	Z2	Z3	Cons	Sel	Src
20	x	y	z	cons	true	FR1

Figure 18: Result with pushing selections to consistent flexible relations.

Some of the cluples are completely lost and other cluples are incomplete since the selection condition is applied before the null value is subsumed by the non-null values. Hence, the above rule is not valid in this case where the source relations are not disjoint but consistent.

Case III: *FR1* and *FR2* are neither disjoint nor consistent.

In this case, the source relations *FR1* and *FR2* may be inconsistent, and also new inconsistencies may result following the union of these relations.

Consider the following example of flexible relations *FR1* and *FR2,* in Figure 19, which are mutually inconsistent.

FR1

K	Z1	Z2	Z3
10	x	y	z
20	u	v	z
30	x	p	v

FR2

K	Z1	Z2	Z3
10	x	y	null
20	x	y	w
40	x	s	z

Figure 19: Inconsistent flexible relations.

Similar to the previous case, consider a query $\sigma^\circ{}_q(FR1 \cup^\circ FR2)$ where $q = [Z2 = y \wedge Z3 = z]$ is the selection predicate. The result obtained without pushing the selection is shown in Figure 20.

$$\sigma_q^\circ(FR1 \cup^\circ FR2)$$

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	y	z	cons	true	FR1
10	x	y	null	cons	true	FR2
20	u	v	z	incons	maybe	FR1
20	x	y	w	incons	maybe	FR2

Figure 20: Result without pushing selections to inconsistent flexible relations.

The result with pushing selections to the source relations is shown in Figure 21.

$$\sigma_q^\circ(FR1) \cup^\circ \sigma_q^\circ(FR2)$$

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	y	z	cons	true	FR1

Figure 21: Result with pushing selections to inconsistent flexible relations.

Once again, an incomplete set of results is obtained by pushing the selection operator. Some tuples are missing from the cluple with $K = 10$ and the cluple with $K = 20$ has been completely eliminated from the result. Hence, the above rule is also not valid in this case where the source relations are neither disjoint nor consistent.

6.2 Strategies for pushing selection over unions

As shown in the previous section, the rule for pushing selections over unions of flexible relations, in its present form, is valid only for the case where the relations are disjoint.

The simplest and potentially expensive solution is not to push selections while materializing flexible relations from its source relations for the latter two cases. Another solution, as discussed below, is to use the notion of a reduction operation over the source relations, analogous to the semi-join operation, to enable a more efficient execution of queries over unions of flexible relations.

In the rest of this section, the semantics of reduction operations for Cases II and III as defined above are discussed.

- *FR1* and *FR2* are not disjoint but consistent (Case II)

  Consider the example of relations *FR1* and *FR2* in Figure 16. The problem with pushing selections in this case is the presence of null values in the source relations. Such null values result in the selection predicate evaluating to *false*. For example, the cluple with $K = 10$ is not selected from the source relations since the value for one of the attributes in the selection predicate is *null* (attribute $Z3$ in relation *FR1* and attribute $Z2$ in relation *FR2*) for this cluple. Hence, a possible reduction operation is to retrieve cluples from the source relations by applying only those selection conditions from the selection predicate where the attribute has a non-null value. Thus, while applying the selection predicate $q = [Z2 = y \wedge Z3 = z]$, over cluple

with *K = 10* in relation *FR1*, only the condition [*Z2 = y*] is applied to this cluple. The condition [*Z3 = z*] is not applied since attribute *Z3* has a null value.

The set of cluples obtained by applying this reduction operation to each of the source relations and merging the results is shown in Figure 22.

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	null	null	cons	true	FR1
10	null	y	null	cons	true	FR1
10	x	null	z	cons	true	FR2
20	x	y	z	cons	true	FR1
20	x	null	z	cons	true	FR2
30	x	null	z	cons	true	FR1

Figure 22: Result with the reduction operation.

Note that tuple ⟨*30, x, p, null*⟩ from *FR2* is rejected since the predicate [*Z2 = y*] evaluates to *false* for this tuple. The final result, obtained by applying the original selection predicate to this intermediate result is shown in Figure 23.

K	Z1	Z2	Z3	Cons	Sel	Src
10	x	null	null	cons	true	FR1
10	null	y	null	cons	true	FR1
10	x	null	z	cons	true	FR2
20	x	y	z	cons	true	FR1
20	x	null	z	cons	true	FR2

Figure 23: Result after applying the original selection predicate.

As shown above, tuple ⟨*30, x, null, z*⟩ selected by the reduction operation, does not satisfy the original selection predicate and is rejected from the final result. The above algorithm is summarized in the following steps.

1) Apply the reduction operation to each source relation.
2) Union the results obtained in Step 1.
3) Apply the original selection predicate to the result of Step 2 to obtain the final result.

Thus, while some extra tuples may be retrieved by the reduction operation, still the overall operation is expected to be more efficient as compared to materializing the entire flexible relation before applying the selection predicate.

• *FR1* and *FR2* are neither disjoint nor consistent (Case III)

Consider the example relations *FR1* and *FR2* in Figure 19. Besides the problem of losing cluples because of null values, as in the previous case, there is also the problem of inconsistent data.

For example, the selection predicate $q = [Z2 = y \wedge Z3 = z]$ evaluates to *false* for each of the cluples with *K = 20* in *FR1* and *FR2*, but the evaluation status of the predicate *q* for cluple with *K = 20* formed by taking a union of these two cluples is *maybe*. Hence, in this case a possible reduction operation over the source relations is to select a cluple from the source relations if it satisfies at least one of the conditions of the selection predicate. Thus, for a selection predicate of the form $q = p_1 \wedge p_2 \wedge \cdots \wedge p_i$ the selection predicate $q' = p_1 \vee p_2 \vee \cdots \vee p_i$ is applied to

each of the source relations during the reduction operation. Similar to the previous case, the original selection predicate *q* is then applied to the result of this reduction operation to determine the final result. The selection predicate *q'* is weak and it selects cluples that may not belong to the final result. However, the final selection operation removes any such extraneous cluples from the result.

There is still the problem of incomplete cluples. The cluples in the result set may not be complete in the sense that the source relations may contain cluples whose entity-identifying attribute set matches the corresponding attribute set of a cluple in the result but which is not in the result set. Such cluples have to be retrieved and merged with the respective cluples in the result. Hence, the source relations have to be queried again to retrieve such related cluples.

The above algorithm is summarized in the following steps. It requires two accesses of the source relations.
1) Convert *q* to *q'*.
2) Perform query *q'* on each source *(Access 1)*.
3) Merge data from the different sources.
4) Perform query *q* on merged data.
5) Determine the set of entity identifiers for the result.
6) For each source determine the list of entity identifiers in the result (Step 5) not received from that source.
7) Fetch all cluples corresponding to these lists from the source relations *(Access 2)*.
8) Merge the cluples obtained from the previous step (Step 7) with the intermediate result from Step 5 and reapply the selection predicate *q* to obtain the final result.

Note that given a set of sources, it is usually not possible to distinguish between cases II and III a priori. Hence it is always necessary to access the source relations twice for processing the query.

Thus, the presence of inconsistencies in the source relations raises new issues for flexible relation query optimization. Application of the techniques developed in the context of classical relation query optimization to the flexible relation case is not expected to yield the desired results. New strategies, similar to those proposed above, need to be considered while optimizing such flexible relation queries.

7. Discussion

This paper is motivated by the need to deal with inconsistencies while manipulating data in sets of autonomous databases. The fundamental assumption in the classical relational model is that the databases are consistent, hence it does not provide any support for dealing with any inconsistencies. Using the view definition mechanisms, as developed in the context of classical relations, it is difficult to define and maintain integrated views in the presence of inconsistent data. As noted earlier, it is difficult, if not impossible to detect and resolve all inconsistencies before defining such views. Also, all the semantics for detecting, representing, and manipulating inconsistent data have to be encoded within the applications.

The flexible relational model, as proposed in this paper, provides support for dealing with inconsistent data. This model enables the definition of integrated views over relations that may be inconsistent. The notion of cluples enables the representation of any sets of inconsistent data obtained upon materializing this view. The flexible relational algebra provides a set of semantics for database operations in the presence of inconsistent data. Thus, the flexible relational

model provides semantics for detecting, representing, and manipulating any inconsistent data derived from multiple source relations.

Issues related to optimizing flexible relation queries in the presence of inconsistent data were considered in Section 6. These issues also reflect the difficulty faced by an application developer while manipulating data in the presence of inconsistent data. Without the flexible relation model, there is currently no notion of query optimization in the presence of inconsistent data. Hence, all such issues have to be considered by the application developer while encoding applications in an autonomous database environment. In practice, they are likely to be ignored and the poor performance of the system will be viewed as a weakness of relational database technology.

Thus, the flexible relational model serves as a framework, both for discovering and also addressing the problems of manipulating data located in distributed, and possibly inconsistent databases.

Acknowledgments

References

[Agarwal92], Agarwal, S.S., *Flexible Relation: A Model for Data in Distributed, Autonomous and Heterogeneous Databases*, Ph.D. Thesis, Department of Electrical Engineering, Stanford University, June 1992.

[Barbara91] Barbara, D., and Garcia-Molina, H., *The demarcation protocol: a technique for maintaining arithmetic constraints in distributed database systems*, CS-TR-320-91, Princeton University, April 1991.

[Bernstein81] Bernstein, P. A., and Goodman, N., "Concurrency Control in Distributed Database Systems," *ACM Computing Surveys*, Vol. 13, No. 2, 1981.

[Breitbart90] Breitbart, Y., Garcia-Molina, H., Litwin, W., Roussopoulos, N., Rusinkiewicz, M., Thompson, G. and Wiederhold, G., "Final Report of the Workshop on Multidatabases and Semantic Interoperability", Tulsa, Oklahoma, November 2-4, 1990.

[Bright92] Bright, M.W., Hurson, A.R., and Pakzad, S.H., "A Taxonomy and Current Issues in Multidatabase Systems," *Computer*, pp. 50-59, March 1992.

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

[Ceri92] Ceri, S., Houstma, M.A.W., Keller, A.M., and Samarati, P., "Independent Updates And Incremental Agreement in Replicated Databases," *Computer Science Technical Report STAN-CS-92-1446*, Stanford University, October 1992.

[Chatterjee91] Chatterjee, A., and Segev, A., "Data Manipulation in Heterogeneous Databases," *SIGMOD Record*, Vol. 20, No. 4, December 1991.

[Codd79] Codd, E.F., "Extending the Database Relational Model to Capture More Meaning," *ACM Transactions on Database Systems*, Vol. 4, No. 4, pp. 397-434, 1979.

[Date83], Date, C.J., *An Introduction to Database Systems*, Vol. 2, Addison-Wesley, Reading, MA, 1983.

[DeMichiel89] DeMichiel, L.G., "Resolving Database Incompatibility: An Approach to Performing Relational Operations over Mismatched Domains," IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 4, December 1989.

[Gamal-Eldin88] Gamal-Eldin, M.S., Thomas, G. and Elmasri, R., "Integrating Relational Databases with Support for Updates," *Proceedings of International Symposium on Databases in Parallel and Distributed Systems*, Austin, Texas, pp. 202-209, 1988.

[Gray81] Gray, J.N., "The Transaction Concept: Virtues and Limitations," *Seventh VLDB*, Cannes, 1981.

[Keller85] Keller, A.M., and Wilkins M.W., "On the Use of an Extended Relational Model to Handle Changing Incomplete Information," *IEEE Transactions on Software Engineering,* SE-11:7, pp. 620-633, July 1985.

[Litwin89] Litwin, W., Abdellatif, A., Zeroual, A., and Nicolas, B., "MSQL: A Multidatabase Language," *Information Sciences* 49, pp. 59 - 101,1989.

[Litwin90] Litwin, W., Mark, L., and Roussopoulos, N., "Interoperability of Multiple Autonomous Databases," *ACM Computing Surveys*, Vol.22, No. 3, September, 1990.

[Maier83] Maier, D., "The Theory of Relational Databases," Computer Science Press, 1983.

[Ramarao88], Ramarao, K.V.S., "Transaction Atomicity in the Presence of Network Partitions," *IEEE Data Engineering Conference*, No. 4, Los Angeles, February 1988.

[Scheuermann94] Scheuermann, P., and Chong, E. I., "Role-based Query Processing in Multidatabase Systems," *EDBT' 94*, Cambridge, England.

[Sheth90] Sheth, A.P., and Larson, J.A., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," *ACM Computing Surveys*, Vol. 22, No. 3, September 1990.

[Traiger82] Traiger, I.L., et al., "Transactions and Consistency in Distributed Database Management Systems," *ACM Transactions on Database Systems,* Vol. 7, No. 3, 1982.

[Wiederhold90] Wiederhold, G., and Qian, X., "Consistency control of replicated data in federated databases," in *Proceedings of the Workshop on Management of Replicated Data*, pp. 130-132, Houston, November 1990.

[Zaniolo84] Zaniolo, C., "Database Relations with Null Values," *Journal of Computer and System Sciences*, 28, pp 142-166, 1984.