Entity Join Optimization in Multidatabase Systems *

Pauray S.M. Tsai and Arbee L.P. Chen

Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 300, R.O.C.

Abstract

Heterogeneities exist in a multidatabase environment. For example, a real world entity may be differently represented in relations of different databases. In particular, keys of these relations may be incompatible. In this paper, we develop an entity join operator named EJ operator, which can be used to join two relations on their compatible/incompatible keys. By this join, if an entity is represented in both relations, all its properties given in relations can be obtained. Besides, a neighborhood join operator named NJ operator is introduced, which joins relations allowing some degree of value uncertainty on the join attributes. The EJ operator, NJ operator and the previously proposed extended relational operators can be combined to support a more powerful set of operations for a multidatabase system. We consider processing entity join queries in a wide area multidatabase environment where the query processing cost is dominated by the cost of data transmission. Since the EJ operator "integrates" tuples from relations, whose attribute values may have conflicts, a natural way to process the operator is to transmit both relations to a site, resolve the conflicts and process the join, which is very costly. An approach which transforms a global query into local subqueries to preprocess queries with EJs in multiple sites is thus proposed with an attempt to lower the cost of data transmission.

1 Introduction

Because of the increasing need for data sharing among multiple databases, the development of multidatabase systems [ASD91, SBD81] has become an important research issue [Br90]. At present, many research results about heterogeneous DBMSs are concerned with the problems of schema integration and interoperability, while very few papers discuss query optimization. The difficulty of query optimization for the heterogeneous DBMSs lies in the complexity of heterogeneities existing among the component databases. However, query optimization is a significant research problem, which greatly affects the system performance.

Many distributed query optimization algorithms have been proposed. Some use semijoins to reduce the cost of data transmission [AHY83, BGW81, CL85, WCS92], which can minimize the query response time or total time. Others consider decomposing a global query into subqueries and processing these subqueries at the sites which contain the referenced data [Ch90, Che90, St75]. This method increases query processing parallelism and therefore reduces the query response time. None of the above query optimization algorithms consider the problem of data inconsistency.

*Appeared in Proc. SIXTH International Conference on Management of Data (1994).

Lu et al. [LOG92] discussed the difference of query optimization problems between heterogeneous distributed database systems and homogeneous distributed database systems. Du et al. [DKS92] proposed a query optimization strategy to resolve the heterogeneities of component query optimizers in heterogeneous DBMSs. In [DKS92], the data and schema conflicts were ignored.

Chatterjee and Segev [CS91] proposed the E–Join operator to join relations in different databases. Issues on missing data or domain mismatch [Br90] which may exist between the two relations to be joined, however, were not considered. The optimization method for the E–Join operator was not considered either. In [TC93], we extended our results on probabilistic query processing [TCY93] to consider joining two relations on their incompatible keys. A new approach was proposed to identify the same entities from different relations considering various data and schema conflicts.

In this paper, we formally define the entity join operator named EJ operator, which joins two relations on their compatible/incompatible keys. An idea of *constraint relaxation* is proposed to achieve the purpose of local processing. Some equivalence transformation rules are also developed to transform the query with EJs into local subqueries such that the size of the relations to be joined can be reduced before the EJ operation is really performed. Besides, we propose a neighborhood join operator named NJ operator, which is used to join two relations on their nonkey attributes. The NJ operation joins relations allowing some degree of value uncertainty on the join attributes. Due to possible value conflicts in different databases, value uncertainty exists in a multidatabase system. The NJ operator is thus proposed to handle this uncertainty.

This paper is organized as follows. In Section 2, we define the EJ operator. In Section 3, the local processing for queries with EJs is considered. Transformation rules are developed to transform a global query into local subqueries to reduce the size of relations to be joined. The NJ operator is introduced in Section 4. Finally, we conclude with the future work in Section 5.

2 EJ Operator

Traditional join operators is not sufficient to identify the same entities from relations in a multidatabase system because various data conflicts may cause them to fail. In this section, we define the EJ operation for joining relations with various data conflicts. The EJ operation uses the probabilistic technique and the concept of *probabilistic partial value* [TCY93] to identify and integrate the same entities in different databases. A probabilistic partial value is a set of possible values with a probability assigned to each possible value, in which exactly one possible value is the true value. We denote a probabilistic partial value as $[u_1^{x_1}, u_2^{x_2}, ..., u_j^{x_j}]$, where $\{u_1, u_2, ..., u_j\}$ is the set of possible values, x_m is the associated probability of u_m and $\sum_{m=1}^{j} x_m = 1$. Assume null values, denoted '$\sim$', may exist in a database. The null value considered is an *applicable null value* as defined in [Co86]. An applicable null value denotes that the value is presently unknown, but can be entered to the database once it is known. Note that a definite value v can be considered as a probabilistic partial value $[v^1]$, and a null value can be expressed as a probabilistic partial value $[v_1^{\frac{1}{n}}, v_2^{\frac{1}{n}}, ..., v_n^{\frac{1}{n}}]$ assuming the possible values have the same probability to be the true value, where $\{v_1, v_2, ..., v_n\}$ is the attribute domain.

Let R_1 and R_2 be two relations, t_1 be a tuple of R_1 and t_2 a tuple of R_2. The EJ operation on relations R_1 and R_2 is denoted as $R_1 \overset{\bullet}{\bowtie} R_2$. There are two cases to be considered:

1. the keys of R_1 and R_2 are compatible

We say the key attributes in relations R_1 and R_2 are compatible if the domains of the key attributes are semantically equivalent. There may be a representation conflict in the domains of compatible key attributes. In the case, a conversion function can be defined to map each domain value of one key attribute to a unique domain value of the other one. The expression $v_2 = map(v_1)$ (or $v_1 = map(v_2)$) indicates that v_1 and v_2 are values in two compatible key attributes and they represent the same entity. Let the schema of R_1 be (K_1, C, X) and that of R_2 be (K_2, C, Y), where K_1 and K_2 are the keys of R_1 and R_2, respectively, which are compatible, C is the set of the common attributes of R_1 and R_2, and X and Y are the remaining attribute sets of R_1 and R_2, respectively. Nonkey attribute a_1 of relation R_1 is *common* to nonkey attribute a_2 of relation R_2 if the domains of a_1 and a_2 are semantically equivalent. In this paper, common attributes are assumed having the same domain. Note that two tuples in R_1 and R_2 representing the same entity may have conflicting values in a common attribute. We use the technique of probabilistic partial values to handle this value conflict in the definition of EJ operation. Let $prob_x(e)$ be the probability of e in set x. The EJ operation on $R_1(K_1, C, X)$ and $R_2(K_2, C, Y)$ produces a relation with the schema (K, C, X, Y), where K is assumed to be K_1. The EJ operation is defined as follows.

$$
\begin{aligned}
&R_1 \overset{\bullet}{\bowtie} R_2 \equiv \\
&\quad \{t | (\exists t_1)(\exists t_2)(t_1 \in R_1 \wedge t_2 \in R_2 \wedge t.K = t_1.K_1 = map(t_2.K_2) \\
&\qquad \wedge (\forall S)(S \in C \wedge t.S = (t_1.S \cup t_2.S) \\
&\qquad\qquad \wedge (\forall e)(e \in t.S \wedge prob_{t.S}(e) = \frac{1}{2}(prob_{t_1.S}(e) + prob_{t_2.S}(e)))) \\
&\qquad \wedge t.X = t_1.X \wedge t.Y = t_2.Y)\}
\end{aligned}
$$

In the above definition, if the key value of tuple t_1 in R_1 is mapped to the key value of tuple t_2 in R_2, then t_1 and t_2 represent the same entity and are joined. The conflicting values v_1 and v_2 for the common attribute in t_1 and t_2, respectively, is represented as a probabilistic partial value $[v_1^{\frac{1}{2}}, v_2^{\frac{1}{2}}]$ by assuming v_1 and v_2 have equal probabilities to be the true value. This operation is a specialization of the Integration operator in [TCY93].

2. the keys of R_1 and R_2 are incompatible
Consider relations $R_1(K_1, C, A_1, X)$ and $R_2(K_2, C, A_2, Y)$ where the keys K_1 and K_2 are incompatible, namely, there is no mapping between the domains of K_1 and K_2 to determine whether two tuples from R_1 and R_2 represent the same entity. A_1 and A_2 are two *related* but not common attribute sets to be described in the following. Assume A_1 and A_2 are $\{a_i\}$ and $\{a_j\}$, respectively. Let the domains of a_i and a_j be $\{u_1, u_2, ..., u_r\}$ and $\{v_1, v_2, ..., v_m\}$, respectively. Assume the relationship between the domains of a_i and a_j is expressed as a *domain hierarchy* as depicted in Figure 1, where the domain of u_1 is $\{v_1, v_2, ..., v_m\}$, the domain of u_2 is $\{w_1, w_2, ..., w_p\}$, ..., and the domain of u_r is $\{x_1, x_2, ..., x_q\}$. Sets $\{v_1, v_2, ..., v_m\}$, $\{w_1, w_2, ..., w_p\}$, ... , and $\{x_1, x_2, ..., x_q\}$ are assumed to be mutually exclusive. Since the keys of R_1 and R_2 are incompatible, we compare the values in the common attributes and related attributes to determine whether two tuples in R_1 and R_2 represent the same entity. Let t_1 and t_2 be the tuples of R_1 and R_2, respectively, and $P_{same}(t_1, t_2)$ be the possibility for t_1 and t_2 to represent the same entity as defined in [TC93]. If $P_{same}(t_1, t_2) > 0$, then t_1 and t_2 will be joined. The EJ operation on $R_1(K_1, C, \{a_i\}, X)$ and $R_2(K_2, C, \{a_j\}, Y)$ produces a relation with the schema $(K_1, K_2, C, \{a_i\}, \{a_j\}, X, Y)$. According to the domain hierarchy in Figure 1, the value u_1 of a_i can be specialized to be one of the

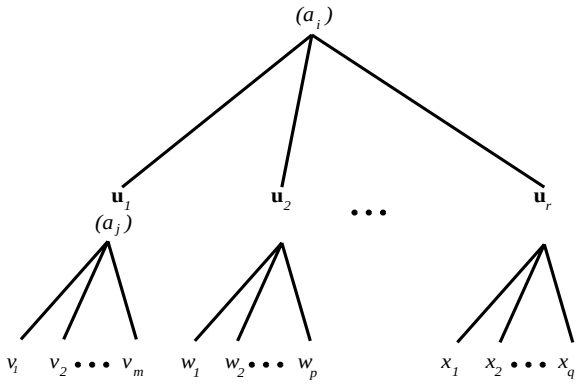

Figure 1: The domain hierarchy.

elements in $\{v_1, ..., v_m\}$. Assume $t_1.a_i$ is u_1, and t_1 and t_2 are to be joined by the EJ operation. The value $t_1.a_i$ needs to be specialized into the probabilistic partial value $[v_1^{\frac{1}{m}}, v_2^{\frac{1}{m}}, ..., v_m^{\frac{1}{m}}]$ such that it can be integrated with $t_2.a_j$. Denote the specialized value as $specialize(t_1.a_i)$. The integrated value becomes the value of a_j for the join tuple t of t_1 and t_2. On the other hand, $t_2.a_j$ needs to be generalized into the value u_1 such that it can be integrated with $t_1.a_i$ to form the value of a_i for the join tuple t. Denote the generalized value as $generalize(t_2.a_j)$. The EJ operation on $R_1(K_1, C, \{a_i\}, X)$ and $R_2(K_2, C, \{a_j\}, Y)$ is defined as follows.

$$
\begin{aligned}
&R_1 \overset{\bullet}{\bowtie} R_2 \equiv \\
&\quad \{t | (\exists t_1)(\exists t_2)(t_1 \in R_1 \wedge t_2 \in R_2 \wedge P_{same}(t_1, t_2) > 0 \wedge t.K_1 = t_1.K_1 \wedge t.K_2 = t_2.K_2 \\
&\qquad \wedge (\forall S)(S \in C \wedge t.S = (t_1.S \cup t_2.S) \\
&\qquad\quad \wedge (\forall e)(e \in t.S \wedge prob_{t.S}(e) = \frac{1}{2}(prob_{t_1.S}(e) + prob_{t_2.S}(e)))) \\
&\qquad \wedge (t.a_i = t_1.a_i \cup generalize(t_2.a_j) \wedge \\
&\qquad\quad (\forall e)(e \in t.a_i \wedge \\
&\qquad\quad prob_{t.a_i}(e) = \frac{1}{2}(prob_{t_1.a_i}(e) + prob_{generalize(t_2.a_j)}(e)))) \\
&\qquad \wedge (t.a_j = t_2.a_j \cup specialize(t_1.a_i) \wedge \\
&\qquad\quad (\forall e)(e \in t.a_j \wedge \\
&\qquad\quad prob_{t.a_j}(e) = \frac{1}{2}(prob_{t_2.a_j}(e) + prob_{specialize(t_1.a_i)}(e)))) \\
&\qquad \wedge t.X = t_1.X \wedge t.Y = t_2.Y)\}
\end{aligned}
$$

The above definition can be easily extended to the cases where there exist more than one related attribute or different types of domain hierarchies such as the one shown in Figure 2. Besides, we can extend the EJ to α-EJ which only produces the join tuples of t_1 and t_2 with $P_{same}(t_1, t_2) \geq \alpha$, where $0 < \alpha \leq 1$. The definition of α-EJ is the same as that of EJ except that $P_{same}(t_1, t_2) > 0$ is replaced by $P_{same}(t_1, t_2) \geq \alpha$.

3 Local Processing for Queries with EJs

In this section, we consider local processing for queries with EJs. Since inconsistent data may exist in different databases, cares need to be taken to transform queries for the local

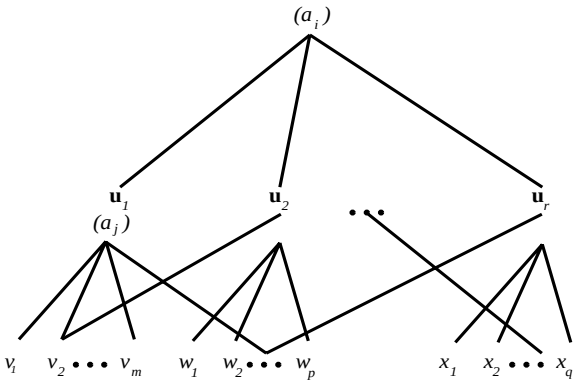

Figure 2: More complex domain hierarchy.

processing. For the sake of simplicity, we only consider queries involving two relations. It can be easily extended to process queries involving more than two relations.

The queries considered are of the form $\sigma_P(R_1 \overset{\bullet}{\bowtie} R_2)$, where R_1 and R_2 represent relations in different databases and P represents the conjunction of a set of simple selection predicates $P_1, P_2, ..., P_n$. A *simple selection predicate* is defined as one of the form *attr* **op** C, where *attr* represents a relation attribute, **op** denotes an operator such as ">," "<" or "=," and C represents a constant.

Let S_p be the set $\{P_1, P_2, ..., P_n\}$. The *associated attribute* for predicate P_i is the *attr* component of P_i. Assume the associated attributes for predicates $P_1, ..., P_i, ..., P_n$ are $a_1, ..., a_i, ..., a_n$, respectively. An associated attribute a_i is called a *private attribute* for relation R_k, where $k = 1, 2$, if a_i appears in R_k and is not related or common to any attribute of the other relation.

Four sets of predicates which will be used in the query transformation in the following sections are defined as follows:
$S_1 = \{P_i | P_i \in S_p$ and a_i is a private attribute for $R_1\}$
$S_2 = \{P_i | P_i \in S_p$ and a_i is a private attribute for $R_2\}$
$S_c = \{P_i | P_i \in S_p$ and a_i is a common attribute for R_1 and $R_2\}$
$S_r = \{P_i | P_i \in S_p$ and a_i is a related attribute for R_1 and R_2 $\}$
The predicates in S_1 and S_2 are called *private selection predicates* for R_1 and R_2, respectively, and predicates in S_c and S_r are called *common selection predicates* for R_1 and R_2, and *related selection predicates* for R_1 and R_2, respectively.

3.1 Motivation

TC

id	*name*	*age*	*class*	*phone*
2	John	21	junior	4335
30	Mary	20	sophomore	4863
41	Jane	22	senior	3776
46	Paul	18	freshman	2375
53	John	22	senior	6447
55	Tony	19	freshman	6450

TW

id	*name*	*age*	*dept*	*class*
30	Mary	20	CS	junior
43	Joe	23	CE	senior
46	Paul	18	CS	freshman
55	Tony	20	CS	sophomore
57	Joe	19	CS	freshman

Figure 3: Relations **TC** and **TW** in two different databases.

Consider the relations in Figure 3, where relation **TC** in one database records the data for students who belong to the tea club, and relation **TW** in another database records the data for students who take Technical Writing. The key attributes *id* in both relations are assumed compatible. A query "Find all twenty-year-old sophomores who belong to the tea club and take Technical Writing" can be expressed as

$$\sigma_{(age=20)\wedge(class=sophomore)}(\mathbf{TC} \overset{\bullet}{\bowtie} \mathbf{TW}).$$

The query cannot be transformed into $(\sigma_{(age\ =\ 20)\ \wedge\ (\ class\ =\ sophomore\ )}\ \mathbf{TC}\ ) \overset{\bullet}{\bowtie} (\sigma_{(age\ =\ 20)\wedge(class\ =\ sophomore)}\ \mathbf{TW})$, even though *age* and *class* are common attributes of relations **TC** and **TW**. Observe the tuple (30, Mary, 20, sophomore, 4863) in **TC** and the tuple (30, Mary, 20, CS, junior) in **TW**. These two tuples represent the same entity, but their *class* values are inconsistent. The entity represented by these two tuples can be considered to be the integrated tuple (30, Mary, 20, CS, [sophomore$^{1/2}$,junior$^{1/2}$], 4863) as shown in Figure 4, and it is a possible answer tuple for the original query as shown in Figure 5. The column *poss* in Figure 5 denotes the possibility of a tuple to be an answer tuple. However, the query result obtained by executing the transformed query $(\sigma_{(age=20)\wedge(class=sophomore)}\mathbf{TC}) \overset{\bullet}{\bowtie} (\sigma_{(age=20)\wedge(class=sophomore)}\mathbf{TW})$ is empty. Therefore, the transformed query is not equivalent to the original one because some tuples possibly satisfying the original query condition are eliminated in the process of local processing.

The situations for a query to be correctly transformed into another one which is applicable for local processing in heterogeneous databases are discussed in the following.

3.2 EJ with Compatible Keys

Consider the query $\sigma_{P_1\wedge P_2\wedge\ldots\wedge P_n}\ (R_1 \overset{\bullet}{\bowtie} R_2)$, where R_1 and R_2 are in different databases and the keys of R_1 and R_2 are compatible. A predicate is called a *locally executable predicates* (LEP) for relation $R_k, k = 1, 2$, if it can be locally processed on R_k without affecting the correctness of the query result.

Transformation 1: The query $\sigma_{P_1\wedge P_2\wedge\ldots\wedge P_n}(R_1 \overset{\bullet}{\bowtie} R_2)$ can be equivalently transformed into

$$\sigma_{\wedge(S_c\cup S_r)}((\sigma_{\wedge(S_1)}R_1) \overset{\bullet}{\bowtie} (\sigma_{\wedge(S_2)}R_2))$$

where $\wedge(S_c \cup S_r)$, $\wedge(S_1)$ and $\wedge(S_2)$ denote the conjunction of predicates in sets $S_c \cup S_r$, S_1 and S_2, respectively.

id	*name*	*age*	*dept*	*class*	*phone*
30	Mary	20	CS	[sophomore$^{1/2}$,junior$^{1/2}$]	4863
46	Paul	18	CS	freshman	2375
55	Tony	[19$^{1/2}$, 20$^{1/2}$]	CS	[freshman$^{1/2}$, sophomore$^{1/2}$]	6450

Figure 4: The result of $\mathbf{TC} \overset{\bullet}{\bowtie} \mathbf{TW}$.

It is obvious that private selection predicates for R_1 and R_2 are LEPs. The common selection predicates or related selection predicates for R_1 and R_2, however, cannot be locally processed since inconsistent data may exist in R_1 and R_2. Therefore, they are processed only after joining $\sigma_{\wedge(S_1)}R_1$ and $\sigma_{\wedge(S_2)}R_2$.

Reconsider the query $\sigma_{(age=20)\wedge(class=sophomore)}(\mathbf{TC} \overset{\bullet}{\bowtie} \mathbf{TW})$ for relations in Figure 3. Since selection attributes *age* and *class* both are common attributes of these two relations, no transformation for local query processing can be performed.

id	*name*	*age*	*dept*	*class*	*phone*	*poss*
30	Mary	20	CS	[sophomore$^{1/2}$, junior$^{1/2}$]	4863	0.5
55	Tony	[19$^{1/2}$, 20$^{1/2}$]	CS	[freshman$^{1/2}$, sophomore$^{1/2}$]	6450	0.25

Figure 5: The result of $\sigma_{(age=20)\wedge(class=sophomore)}(\mathbf{TC} \overset{\bullet}{\bowtie} \mathbf{TW})$.

3.3 EJ with Incompatible Keys

We consider local query processing involving EJs with incompatible keys in this section.

Teacher

id	*name*	*department*	*degree*	*age*
7001	Mary	CS	∼	30
7002	Paul	EE	PhD	29
7003	John	CS	PhD	29
7004	John	CS	MS	26
7005	Lina	EE	PhD	34
7006	Paul	CS	PhD	30

Consultant

id	*name*	*specialty*	*degree*	*age*	*city*
101	John	CN	MS	25	B
102	Jose	DB	MS	36	∼
103	James	DB	BS	24	A
104	Mary	CN	MS	29	B
105	John	AI	MS	26	B
106	Dick	DB	BS	24	A
107	Tony	IP	MS	28	∼
108	Paul	DB	MS	25	B
109	Mary	AI	PhD	30	B
110	Paul	DB	PhD	35	∼

Figure 6: Relations **Teacher** and **Consultant** in two different databases.

Consider the relations in Figure 6, where relation **Teacher** in one database records the personal data of teachers at X University, and relation **Consultant** in another database records the data of consultants in the computer science division of Y Company. The key attribute *id* in **Teacher** represents the identification number for teachers at X University while that in **Consultant** represents the identification number for consultants at Y Company. Obviously, the keys in **Teacher** and **Consultant** are incompatible.

Consider the query $\sigma_{P_1\wedge P_2\wedge\ldots\wedge P_n}(R_1 \overset{\bullet}{\bowtie} R_2)$, where the keys of R_1 and R_2 are incompatible. **Transformation 1** introduced in Section 3.2 also applies for the case here. Let t_1 be a tuple of R_1 and t_2 a tuple of R_2. Assume a_i is the associated attribute of predicate P_i. In the following, we say a join tuple t (i.e., a tuple exists in the join result) may satisfy P_i if there is a possible value in the probabilistic partial value $t.a_i$, which satisfies predicate P_i. Three rules derived from [TC93] useful for locally preprocessing a query are described as follows.

1. In some cases, we do not allow the values of a common attribute in t_1 and t_2 to be inconsistent if t_1 and t_2 are considered the same entity. We call such a common attribute a *dominant attribute*. Conversely, a common attribute is called a *nondominant attribute* if the values of the common attribute in t_1 and t_2 are allowed to be different when these two tuples are considered the same entity. For example, attribute *name* in Figure 6 can be considered to be a dominant attribute, and tuples (7001, Mary, CS, ∼, 30) and (101, John, CN, MS, 25, B) in **Teacher** and **Consultant**, respectively, are thus considered representing different entities.

   Rule 1: If one of the dominant attributes has different values in t_1 and t_2, t_1 and t_2 are considered representing different entities.

Transformation 2: By Rule 1, if the associated attribute of common selection predicate P_i is a dominant attribute, query $\sigma_{P_1 \wedge \ldots \wedge P_i \wedge \ldots \wedge P_n}(R_1 \overset{\bullet}{\bowtie} R_2)$ can be equivalently transformed into

$$\sigma_{P_1 \wedge \ldots \wedge P_{i-1} \wedge P_{i+1} \wedge \ldots \wedge P_n}((\sigma_{P_i} \quad R_1) \overset{\bullet}{\bowtie} (\sigma_{P_i} \quad R_2)).$$

This is because if t_1 and t_2 are considered to represent the same entity and their join tuple satisfies the query condition, both of them must satisfy P_i. Therefore, predicates whose associated attributes are dominant are *LEP*s.

2. Consider a nondominant attribute a_i. Let the values of a_i in t_1 and t_2 be v_1 and v_2, respectively, the *difference distance* between v_1 and v_2, denoted $DD(v_1, v_2)$, be a value representing the difference between v_1 and v_2, and the *maximum difference distance* for attribute a_i, denoted $MDD(a_i)$, be a value which $DD(v_1, v_2)$ cannot exceed if t_1 and t_2 are considered to represent the same entity. For example, the difference distance of any two values v_1 and v_2 in attribute *age* can be defined as $|v_1 - v_2|$. Similarly, for nonnumeric attributes such as *city*, we can define the difference distance of any two values in the attribute domain by the property of the attribute, for instance, the relative distance between any two cities in the domain of *city*. The difference distance of attribute values and the maximum difference distance for an attribute can be defined by the database administrator and stored in a domain knowledge base. If the maximum difference distance of an attribute is zero, the attribute is considered a dominant attribute.

Rule 2: If the difference distance between values of $t_1.a_i$ and $t_2.a_i$ is greater than $MDD(a_i)$, then the two entities represented by t_1 and t_2 are different.

Assume P_i is a common selection predicate, and the associated attribute of P_i is the nondominant attribute a_i. Obviously, predicate P_i cannot be a *LEP* for R_1 or R_2. By Rule 2, if tuples t_1 and t_2 are considered representing the same entity, the difference distance between $t_1.a_i$ and $t_2.a_i$ must be less than or equal to $MDD(a_i)$. We consider a relaxed form of P_i to be a *LEP* for R_1 and R_2 in the following, assuming a_i a numerical attribute.

- **case 1**: if P_i represents $a_i = v$, where v is a constant, the relaxed constraint $RC(P_i)$ for P_i is defined as
  $RC(P_i)$: $v - MDD(a_i) \leq a_i \leq v + MDD(a_i)$.
  interpretation: Let the value of attribute a_i in the tuple t_1 of relation R_1 be v'. If v' falls in the range of $[v - MDD(a_i), v + MDD(a_i)]$, then t_1 may be joined with a tuple t_2 in R_2 and the resulting join tuple may satisfy P_i. But if $v' < v - MDD(a_i)$, it is impossible to find a tuple in R_2, which satisfies P_i and represents the same entity as the one represented by t_1. Since if t_2 in R_2 satisfies P_i (i.e., $t_2.a_i = v$), the difference distance between $t_1.a_i$ and $t_2.a_i$ would be greater than $MDD(a_i)$. Finally, if $v' > v + MDD(a_i)$, we cannot find a tuple in R_2, which satisfies P_i and represents the same entity as the one represented by t_1. The reason is the same as that for $v' < v - MDD(a_i)$. Therefore, $RC(P_i)$ is a *LEP* for R_1 and R_2.
- **case 2**: if P_i represents $a_i \geq v$, the relaxed constraint for P_i is defined as
  $RC(P_i)$: $a_i \geq v - MDD(a_i)$.
  Similar to case 1, $RC(P_i)$ can be shown to be a *LEP* for R_1 and R_2.
- **case 3**: if P_i represents $a_i \leq v$, the relaxed constraint for P_i is defined as
  $RC(P_i)$: $a_i \leq v + MDD(a_i)$.
  Similar to case 1, $RC(P_i)$ can be shown to be a *LEP* for R_1 and R_2.

The notion of the relaxed constraint can be easily extended to nonnumerical attributes.

Transformation 3: By Rule 2, if the associated attribute of a common selection predicate P_i is a nondominant attribute, query $\sigma_{P_1 \wedge ... \wedge P_i \wedge ... \wedge P_n}(R_1 \overset{\bullet}{\bowtie} R_2)$ can be equivalently transformed into

$$\sigma_{P_1 \wedge ... \wedge P_i \wedge ... \wedge P_n}((\sigma_{RC(P_i)} \ R_1) \overset{\bullet}{\bowtie} (\sigma_{RC(P_i)} \ R_2)).$$

Since $RC(P_i)$ is a *LEP* for R_1 and R_2, $(\sigma_{RC(P_i)} \ R_1)$ and $(\sigma_{RC(P_i)} \ R_2)$ can be locally processed to reduce the costs of data transmission and join processing. Let R_1' and R_2' be the results of $(\sigma_{RC(P_i)} \ R_1)$ and $(\sigma_{RC(P_i)} \ R_2)$, respectively. If tuples t_1' and t_2' in R_1' and R_2', respectively, are considered representing the same entity, the join result of t_1' and t_2' may not satisfy P_i. For example, in **case 1**, if the values of $t_1'.a_i$ and $t_2'.a_i$ are v_1 and v_2, respectively, where $v < v_1, v_2 \leq v + MDD(a_i)$, then the join tuple of t_1' and t_2' will not satisfy P_i (i.e., the condition $a_i = v$). Therefore, predicate P_i still needs to be evaluated after the EJ operation is processed.

3. Data scaling conflicts may exist in heterogeneous databases. For example, there exists a data scaling conflict between **Teacher**.*department* and **Consultant**.*specialty* in Figure 6. The attributes *department* and *specialty* are considered related attributes and would be used to help identify the same entity. Let a_i and a_j be the related attributes for R_1 and R_2. Assume the relationship between the domains of a_i and a_j is the same as that depicted in Figure 1. Let t_1 and t_2 be tuples in R_1 and R_2, respectively. Consider the following two cases:

   - **case 1**: if a_j is the associated attribute of the related selection predicate P_j

     Rule 3: If $t_1.a_i = u_k$, where $k \neq 1$, t_1 and t_2 are considered representing different entities.

     Transformation 4: By Rule 3, query $\sigma_{P_1 \wedge ... \wedge P_j \wedge ... \wedge P_n}(R_1 \overset{\bullet}{\bowtie} R_2)$ can be equivalently transformed into

     $$\sigma_{P_1 \wedge ... \wedge P_j \wedge ... \wedge P_n}((\sigma_{(a_i = u_1)} \ R_1) \overset{\bullet}{\bowtie} \ R_2).$$

     Predicate P_j cannot be a *LEP* for R_2. Since the tuple in $(\sigma_{(a_i=u_1)} \ R_1)$ can be considered having the value of a_j to be $[v_1^{\frac{1}{m}}, v_2^{\frac{1}{m}}, ..., v_m^{\frac{1}{m}}]$, the join tuples of $(\sigma_{(a_i=u_1)} \ R_1) \overset{\bullet}{\bowtie} R_2$ will have positive possibilities to satisfy P_j no matter what value of a_j in t_2 is. For example, consider the query $\sigma_{(specialty=AI)}$ (**Teacher** $\overset{\bullet}{\bowtie}$ **Consultant**). The relationship between *specialty* and *department* is depicted in Figure 7. Assume tuple (7004, John, CS, MS, 26) in relation **Teacher** and tuple (101, John, CN, MS, 25, B) in relation **Consultant** may represent the same entity. Then the join tuple of these two tuples may satisfy the predicate "specialty=AI" because the velue "CS" in the first tuple can be considered the probabilistic partial value $[CN^{\frac{1}{4}}, DB^{\frac{1}{4}}, IP^{\frac{1}{4}}, AI^{\frac{1}{4}}]$ and the join tuple is considered having the value of *specialty* as $[CN^{\frac{5}{8}}, DB^{\frac{1}{8}}, IP^{\frac{1}{8}}, AI^{\frac{1}{8}}]$ according to the definition of EJ operation. The resulting join tuple therefore has possibility "$\frac{1}{8}$" to satisfy predicate "specialty = AI" although the value of *specialty* for the tuple in **Consultant** is "CN".

- **case 2**: if a_i is the associated attribute of the related selection predicate P_i.

  Transformation 5: By Rule 3, if the value u_1 of a_i does not satisfy P_i, then the query result of $\sigma_{P_1 \wedge ... \wedge P_i \wedge ... \wedge P_n}(R_1 \overset{\bullet}{\bowtie} R_2)$ is empty since all the entities represented by tuples in R_2 cannot satisfy P_i.

  Transformation 6: By Rule 3, if the value u_1 of a_i satisfies P_i, then query $\sigma_{P_1 \wedge ... \wedge P_i \wedge ... \wedge P_n}(R_1 \overset{\bullet}{\bowtie} R_2)$ can be equivalently transformed into

  $$\sigma_{P_1 \wedge ... \wedge P_{i-1} \wedge P_{i+1} \wedge ... \wedge P_n}((\sigma_{(a_i = u_1)} \ R_1) \overset{\bullet}{\bowtie} \ R_2).$$

  All the tuples in R_2 implicitly satisfy P_i since all the values of a_j in R_2 can be generalized to u_1. Moreover, only the tuples in R_1 with the value of a_i being u_1 can be considered in the EJ operation. Therefore predicate P_i is a *LEP* for R_1.

3.4 An Example

Now, we give an example to illustrate how a query is transformed into an equivalent one suitable for local processing. Consider the relations in Figure 6 and the query

$$\sigma_{(name=John \vee Paul) \ \wedge \ (department=CS) \ \wedge \ (specialty=AI \vee DB) \ \wedge \ (age \leq 30) \ \wedge \ (city=B)} (\textbf{Teacher} \overset{\bullet}{\bowtie} \textbf{Consultant}).$$

Some needed information for the query transformation is listed as follows:
(1) the set of common selection predicates = {(name = John $\vee$ Paul), (age $\leq$ 30) }
(2) the set of private selection predicates for **Teacher** = {}
(3) the set of private selection predicates for **Consultant** = {(city = B) }
(4) the set of related selection predicates = {(specialty = AI $\vee$ DB, department=CS)}

The notation (specialty = AI $\vee$ DB, department=CS) indicates that predicates "specialty=AI$\vee$DB" and "department=CS" are a pair of related selection predicates because attributes *specialty* and *department* are a pair of related attributes expressed as (specialty,department).

(5) the set of common attributes = {name, degree, age}
(6) the set of related attributes = {(specialty, department)}
(7) the set of dominant attributes = {name}
(8) the set of nondominant attributes = {(specialty, department), degree, age}
(9) For any two values v_1 and v_2 in attribute *age*, the difference distance $DD(v_1,v_2)$ between v_1 and v_2 is defined as

$$DD(v_1, v_2) = \mid v_1 - v_2 \mid.$$

(10) the maximum difference distance for attribute *age* is 2
(11) the domain hierarchy

- The domain of *department* consists of Computer Science(CS) and Electronic Engineering(EE).
- The domain of *specialty* consists of Computer Network(CN), Database(DB), Image Processing(IP), and Artificial Intelligence(AI).
- The domain of Electronic Engineering(EE) consists of Communication Electronics(CE) and IC Design(IC).

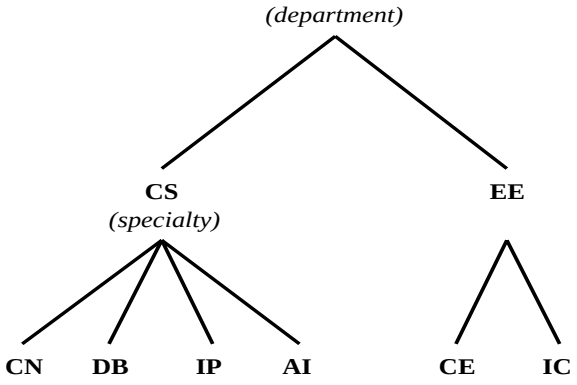

Figure 7: The domain hierarchy for attributes *department* and *specialty*.

The domain hierarchy is depicted in Figure 7. The query transformation is performed as follows:

$\sigma_{(name=John \vee Paul) \wedge (department=CS) \wedge (specialty=AI \vee DB) \wedge (age \leq 30) \wedge (city=B)}$
(**Teacher** $\overset{\bullet}{\bowtie}$ **Consultant**)

$= \sigma_{(name=John \vee Paul) \wedge (department=CS) \wedge (specialty=AI \vee DB) \wedge (age \leq 30)}$
(**Teacher** $\overset{\bullet}{\bowtie}$ ($\sigma_{(city=B)}$ **Consultant**))
(by **Transformation 1**)

$= \sigma_{(department=CS) \wedge (specialty=AI \vee DB) \wedge (age \leq 30)}$
(($\sigma_{(name=John \vee Paul)}$ **Teacher**) $\overset{\bullet}{\bowtie}$ ($\sigma_{(city=B) \wedge (name=John \vee Paul)}$ **Consultant**))
(by **Transformation 2**)

$= \sigma_{(department=CS) \wedge (specialty=AI \vee DB) \wedge (age \leq 30)}$
(($\sigma_{(name=John \vee Paul) \wedge (age \leq 32)}$ **Teacher**) $\overset{\bullet}{\bowtie}$
($\sigma_{(city=B) \wedge (name=John \vee Paul) \wedge (age \leq 32)}$ **Consultant**))
(by **Transformation 3**)

$= \sigma_{(department=CS) \wedge (specialty=AI \vee DB) \wedge (age \leq 30)}$
(($\sigma_{(name=John \vee Paul) \wedge (age \leq 32) \wedge (department=CS)}$ **Teacher**) $\overset{\bullet}{\bowtie}$
($\sigma_{(city=B) \wedge (name=John \vee Paul) \wedge (age \leq 32)}$ **Consultant**))
(by **Transformation 4**)

$= \sigma_{(specialty=AI \vee DB) \wedge (age \leq 30)}$
(($\sigma_{(name=John \vee Paul) \wedge (age \leq 32) \wedge (department=CS)}$ **Teacher**) $\overset{\bullet}{\bowtie}$
($\sigma_{(city=B) \wedge (name=John \vee Paul) \wedge (age \leq 32)}$ **Consultant**))
(by **Transformation 6**)

Figure 8 shows the results of local processing for subqueries
($\sigma_{(name=John \vee Paul) \wedge (age \leq 32) \wedge (department=CS)}$ **Teacher**) and
($\sigma_{(city=B) \wedge (name=John \vee Paul) \wedge (age \leq 32)}$ **Consultant**) in different databases. For the final EJ and selection, refer to [TC93]. Since our purpose is to reduce the cost of data transmission in the wide area multidatabase environment, the developed transformation rules can reduce the size of relations to be join, which are useful to optimize queries with EJs.

4 NJ Operator

In Section 2 we define the EJ operator which can be used to join the tuples representing the same entity from two relations. Because of the existence of inconsistent data in het-

$\sigma_{(name=John \vee Paul) \wedge (age \leq 32) \wedge (department=CS)}$ **Teacher**

id	*name*	*department*	*degree*	*age*
7003	John	CS	PhD	29
7004	John	CS	MS	26
7006	Paul	CS	PhD	30

$\sigma_{(city=B) \wedge (name=John \vee Paul) \wedge (age \leq 32)}$ **Consultant**

id	*name*	*specialty*	*degree*	*age*	*city*
101	John	CN	MS	25	B
105	John	AI	MS	26	B
108	Paul	DB	MS	25	B

Figure 8: The results of ($\sigma_{(name=John \vee Paul) \wedge (age \leq 32) \wedge (department=CS)}$ **Teacher**) and ($\sigma_{(city=B) \wedge (name=John \vee Paul) \wedge (age \leq 32)}$ **Consultant**) in different databases.

erogeneous databases, traditional equijoin operator may not be suitable for joining nonkey attributes as well. In this section, we develop a *neighborhood join* operator named NJ operator, which can be used to join two relations on their nonkey attributes based on a *similarity threshold* specified by the user. That is, if the similarity of the join attribute values for the two tuples is greater than or equal to the similarity threshold, then these two tuples can be joined.

Let R_1 and R_2 be two relations, t_1 be a tuple of R_1 and t_2 a tuple of R_2. The NJ operation on relations R_1 and R_2 is denoted as $R_1 \stackrel{\sim}{\bowtie}^{\alpha}_{R_1.a_1:R_2.a_2} R_2$, where a_1 and a_2 are join attributes and α, $0 < \alpha \leq 1$, is the similarity threshold.

The NJ operation on $R_1(X, a_1)$ and $R_2(Y, a_2)$ has the schema (X, Y, a_1, a_2), where a_1 and a_2 are the join attributes of the NJ operation. Let $sim(t_1.a_1, t_2.a_2)$ be the similarity of $t_1.a_1$ and $t_2.a_2$. The NJ operation is defined as follows.

$$
\begin{aligned}
R_1 \stackrel{\sim}{\bowtie}^{\alpha}_{R_1.a_1:R_2.a_2} R_2 \equiv \\
\{t | (\exists t_1)(\exists t_2)(t_1 \in R_1 \wedge t_2 \in R_2 \wedge sim(t_1.a_1, t_2.a_2) \geq \alpha \\
\wedge t.a_1 = t_1.a_1 \wedge t.a_2 = t_2.a_2 \\
\wedge t.X = t_1.X \wedge t.Y = t_2.Y)\}
\end{aligned}
$$

The neighborhood join operation can also be applied to the case where one joining attribute is a nonkey attribute while the other is a key attribute.

Consider the query of the form $\sigma_P(R_1 \stackrel{\sim}{\bowtie}^{\alpha}_{R_1.a_k:R_2.a_l} R_2)$, where P represents the conjunction of a set of simple selection predicates and a_k and a_l are the join attributes. Since the neighborhood join operation is considered to be the join of tuples in different relations, which need not represent the same entity, the associated attribute for any selection predicate has to be qualified by the name of the corresponding relation to distinguish the attributes in different relations. Let P be $P_1 \wedge ... \wedge P_i \wedge P_{i+1} \wedge ... \wedge P_n$. Assume the associated attributes for predicates $P_1, ..., P_i, P_{i+1}, ..., P_n$ are $R_1.a_1, ..., R_1.a_i, R_2.a_{i+1}, ..., R_2.a_n$, respectively. The query $\sigma_{P_1 \wedge ... \wedge P_n}(R_1 \stackrel{\sim}{\bowtie}^{\alpha}_{R_1.a_k:R_2.a_l} R_2)$ can be equivalently transformed into $((\sigma_{P_1 \wedge ... \wedge P_i} R_1) \stackrel{\sim}{\bowtie}^{\alpha}_{R_1.a_k:R_2.a_l} (\sigma_{P_{i+1} \wedge ... \wedge P_n} R_2))$. It is because that $\{R_1.a_1, ..., R_1.a_i\}$ and $\{R_2.a_{i+1}, ..., R_2.a_n\}$ are considered the sets of private attributes for relations R_1 and R_2, respectively.

4.1 Neighborhood Join with the Same Domain

Consider the neighborhood join $R_1 \overset{\sim}{\bowtie}^{\alpha}_{R_1.a_1:R_2.a_2} R_2$, where attributes a_1 and a_2 have the same domain DOM. The difference distance between any two values v_1 and v_2 in DOM is denoted as $DD(v_1, v_2)$. Let t_1 and t_2 be tuples in R_1 and R_2, respectively. The degree of dissimilarity of $t_1.a_1$ and $t_2.a_2$ is defined as

$$dissim(t_1.a_1, t_2.a_2) = \frac{DD(t_1.a_1, t_2.a_2)}{MAX\{DD(x, y)|x, y \in DOM\}}$$

and the similarity $sim(t_1.a_1, t_2.a_2)$ of $t_1.a_1$ and $t_2.a_2$ is $1 - dissim(t_1.a_1, t_2.a_2)$. Therefore, tuples t_1 and t_2 can be joined for the query $R_1 \overset{\sim}{\bowtie}^{\alpha}_{R_1.a_1:R_2.a_2} R_2$ if $sim(t_1.a_1, t_2.a_2) \geq \alpha$.

Consider the relations **Supplier** and **Customer** in Figure 9 and the query **Supplier** $\overset{\sim}{\bowtie}^{0.8}_{Supplier.city:Customer.city}$ **Customer**, where *Supplier.city* and *Customer.city* have the same domain {Keelung, Taipei, Hsinchu, Taichung, Tainan, Kaohsiung}. The values in

Supplier

s#	*city*
s1	Hsinchu
s2	Kaohsiung
s3	Taichung

Customer

c#	*city*
c1	Taipei
c2	Hsinchu
c3	Taichung
c4	Tainan
c5	Keelung

Figure 9: Relations **Supplier** and **Customer** in two different databases.

the domain of *city* can be ordered by their geographic locations from north to south. They are Keelung, Taipei, Hsinchu, Taichung, Tainan, and Kaohsiung in sequence. The difference distance of any two values in the domain of *city* is defined as the relative distance between these two cities. Assume $DD(Keelung, Taipei)$, $DD(Taipei, Hsinchu)$, $DD(Hsinchu, Taichung)$, $DD(Taichung, Tainan)$ and $DD(Tainan, Kaohsiung)$ are 1, 2, 3, 4, 1, respectively. The difference distance values for the other cases can be evaluated easily. For example, $DD(Keelung, Kaohsiung)$ can be obtained by adding $DD(Keelung, Taipei)$, $DD(Taipei, Hsinchu)$, $DD(Hsinchu, Taichung)$, $DD(Taichung, Tainan)$ and $DD(Tainan, Kaohsiung)$, and its value is 11. Figure 10 shows the result of **Supplier** $\overset{\sim}{\bowtie}^{0.8}_{Supplier.city:Customer.city}$ **Customer**. Because the similarity threshold is 0.8 and $MAX\{DD(x, y)|x, y \in domain(city)\}$ is $DD(Keelung, Kaohsiung)$ whose value is 11, only the tuples t_1 and t_2 in **Supplier** and **Customer**, respectively, with $DD(t_1.city, t_2.city) < 2.2$ can be joined. The column *sim* in Figure 10 denotes the similarity between *Supplier.city* and *Customer.city* for each answer tuple.

s#	*Supplier.city*	*c#*	*Customer.city*	*sim*
s1	Hsinchu	c1	Taipei	0.82
s1	Hsinchu	c2	Hsinchu	1
s2	Kaohsiung	c4	Tainan	0.91

Figure 10: The result of **Supplier** $\overset{\sim}{\bowtie}^{0.8}_{Supplier.city:Customer.city}$ **Customer**.

4.2 Neighborhood Join with Domain Hierarchy

Consider the neighborhood join $R_1 \tilde{\bowtie}^{\alpha}_{R_1.a_i:R_2.a_j} R_2$, where a_i and a_j are the attributes of R_1 and R_2, respectively. Assume the relationship between the domains of a_i and a_j is the same as that depicted in Figure 1. The notation $R_1.a_i : R_2.a_j$ denotes that the values of attribute a_j will be used as the reference for the join. Conversely, the notation $R_2.a_j : R_1.a_i$ in the neighborhood join $R_2 \tilde{\bowtie}^{\alpha}_{R_2.a_j:R_1.a_i} R_1$ denotes that the values of a_i will be used as the reference for the join. Let t_1 and t_2 be tuples in R_1 and R_2, respectively. Two cases are considered as follows.

- **case 1**: $R_1 \tilde{\bowtie}^{\alpha}_{R_1.a_i:R_2.a_j} R_2$
  Since the values of a_j are used as the reference for the join, the values of a_i need to be specialized. Denote the specialized value for $t_1.a_i$ as $specialize(t_1.a_i)$. In our example, if $t_1.a_i$ is u_1, $specialize(t_1.a_i)$ is the probabilistic partial value $[v_1^{\frac{1}{m}}, v_2^{\frac{1}{m}}, ..., v_m^{\frac{1}{m}}]$. The similarity of $t_1.a_i$ and $t_2.a_j$ is defined as

$$sim(t_1.a_i, t_2.a_j) = \frac{1}{2} \sum_{e \in (t_2.a_j \cap specialize(t_1.a_i))} (prob_{t_2.a_j}(e) + prob_{specialize(t_1.a_i)}(e))$$

  where $prob_x(e)$ is the probability of e in set x.

- **case 2**: $R_2 \tilde{\bowtie}^{\alpha}_{R_2.a_j:R_1.a_i} R_1$
  In this case, the values of a_i are used as the reference for the join, therefore, the values of a_j need to be generalized. Denote the generalized value for $t_2.a_j$ as $generalize(t_2.a_j)$. In our example, $generalize(t_2.a_j)$ is u_1. The similarity of $t_1.a_i$ and $t_2.a_j$ is defined as

$$sim(t_1.a_i, t_2.a_j) = \frac{1}{2} \sum_{e \in (t_1.a_i \cap generalize(t_2.a_j))} (prob_{t_1.a_i}(e) + prob_{generalize(t_2.a_j)}(e)).$$

5 Conclusions and Future Work

In this paper, we develop two useful join operators – the EJ operator and the NJ operator, for joining relations in a multidatabase system. The EJ operator can join relations with incompatible keys by using the probabilistic technique to identify the same entities in different relations, while the NJ operator joins tuples having the similarity of their join attribute values greater than or equal to a similarity threshold. These two join operators can be combined with the set of extended relational operators in [TCY93] to support a more powerful set of operations for a multidatabase system. Besides, we consider the local processing to optimize the EJ operation. A set of qualitative equivalence transformation rules are developed to transform a global query into local subqueries such that the size of the relations to be joined can be reduced, which saves the cost of data transmission, especially in the wide area multidatabase environment.

The selection condition considered in this paper is a conjunction of a set of simple selection predicates. However, it can be extended to include disjunctive predicates. The semijoin method can also be extended to further optimize queries with EJs.

References

[ASD91] R. Ahmed, P.D. Smedt, W. Du, W. Kent, M.A. Ketabchi, W.A. Litwin, A. Rafii, and M.C. Shan, The Pegasus Heterogeneous Multidatabase System, *IEEE COMPUTER*, December (1991) pp. 19-27.

[AHY83] P. Apers, A. Hevner, and S.B. Yao, Optimization algorithms for distributed queries, *IEEE Transactions on Software Engineering,* 9 (1), (1983) pp. 57-68.

[BGW81] P. Bernstein, N. Goodman, E. Wong, C. Reeve and J. Rothnie, Query Processing in a System for Distributed Databases (SDD–1), *ACM Transactions on Database Systems,* 6 (4), (1981) pp. 602-625.

[Br90] Y. Breitbart, Multidatabase Interoperability, *SIGMOD RECORD,* 19 (3) (1990) pp. 53-60.

[CS91] A. Chatterjee and A. Segev, Data Manipulation in Heterogeneous Databases, *SIGMOD RECORD,* 20 (4), (1991) pp. 64-68.

[CL85] A.L.P. Chen and V.O.K. Li, An Optimal Algorithms for Processing Distributed Star Queries, *IEEE Transactions on Software Engineering,* 11, (1985) pp. 1097-1107.

[Ch90] A.L.P. Chen, Outerjoin Optimization in Multidatabase Systems, *Proc. IEEE International Symposium on Databases in Parallel and Distributed Systems (DPDS)*, (1990) pp. 211-218.

[Che90] A.L.P. Chen, A Localized Approach to Distributed Query Processing, *Proc. International Conference on Extending Data Base Technology (EDBT)*, (1990) pp.

[Co86] E.F. Codd, Missing Information (Applicable and Inapplicable) in Relational Databases, *SIGMOD RECORD,* 15 (4) (1986) pp. 53-78.

[DKS92] W. Du, R. Krishnamurthy and M.C. Shan, Query Optimization in Heterogeneous DBMS, *Proc. VLDB,* (1992) pp. 277-291.

[LOG92] H. Lu, B.C. Ooi and C.H. Goh, On Global Multidatabase Query Optimization, *SIGMOD RECORD* 21 (4) (1992) pp. 6-11.

[SBD81] J.M. Smith, P.A. Bernstein, U. Dayal, N. Goodman, T. Landers, K.W.T. Lin, and E. Wong, Multibase – Integrating Heterogeneous Distributed Database Systems, *Proceedings of AFIPS NCC,* (1981) pp. 487-499.

[St75] M. Stonebraker, Implementation of Integrity Constraints and Views by Query Modification, *proc. ACM SIGMOD* (1975) pp. 65-78.

[TC93] P.S.M. Tsai and A.L.P. Chen, Querying uncertain data in heterogeneous databases, *Proc. IEEE Third International Workshop on Research Issues on Data Engiennring: Interoperability in Multidatabase Systems* (1993) pp. 161-168.

[TCY93] F.S.C. Tseng, A.L.P. Chen, and W.P. Yang, Answering heterogeneous database queries with degrees of uncertainty, *Distributed and Parallel Databases: an International Journal,* Kluwer Academic Publishers, (1993) pp. 281-302.

[WCS92] C. Wang, A.L.P. Chen and S. C. Shyu, A Parallel Execution Method to Minimizing Distributed Query Response Time, *IEEE Transactions on Parallel and Distributed Systems,* 3 (3), (1992) pp. 325-333.