JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 16, 593-617 (2000)

Partial Natural Outerjoin – An Operation for Interoperability in a Multidatabase Environment

PAURAY S. M. TSAI* AND ARBEE L. P. CHEN
*Department of Information Management
Ming Hsin Institute of Technology
Hsinchu, Taiwan 304, R.O.C.
E-mail: pauray@mis.mhit.edu.tw
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.
E-mail: alpchen@cs.nthu.edu.tw

Natural outerjoin has been considered as an important operation in schema integration. It can be used to define various views in cooperation with other operations. Due to the existence of inconsistent data and null values in base relations of multiple databases, the traditional natural outerjoin cannot be directly applied to schema integration in a multidatabase environment. In this paper, the effects of execution orders, inconsistent data, and null values on the resultant semantics of natural outerjoins are explored. Because an arbitrary execution order of natural outerjoins may cause the resultant semantics to be ambiguous, care needs to be taken in specification of the execution order. We investigate how to determine the execution order of natural outerjoins such that the result is *desirable*. Moreover, an extension of traditional natural outerjoin, called *partial natural outerjoin*, is proposed to handle null values and inconsistent data. When a user issues a query against the global view, the query is modified to obtain one which may contain partial natural outerjoins, selections, and projections, based on the definition of the global view. A set of equivalence transformation rules is developed to transform a modified query into one with simpler operations, which lowers the query processing cost. Moreover, the semijoin technique is applied in query processing. Therefore, the cost of data transmission for processing a query can be further reduced, especially in a wide area network environment.

Keywords: natural outerjoin, multidatabase, inconsistent data, null value, execution order, semantics analysis, equivalence transformation rule

1. INTRODUCTION

Because of the rapid advances in networking technologies and the requirement of data sharing among multiple databases, the development of multidatabase systems [6] has been considered as an important research issue. One of the important characteristics of a multidatabase system is that the autonomy of its component databases is preserved; that is, in a component database, data can be created and manipulated independent of other databases.

In order to provide a high level of transparency and a uniform interface for users to retrieve data in a multidatabase system, the schemas of the component databases are usually integrated to form a global schema. A variety of approaches to data/schema integra-

Received July 28, 1999; revised October 15, 1999; accepted January 3, 2000.
Communicated by Wei-Pang Yang.

tion have been proposed [11, 13]. Batini et al. surveyed twelve methodologies for database or view integration in [2]. In [18] and [22], sets of operators were developed for virtual integration of multiple databases. To resolve conflicts among component schemas, DeMichiel [14] proposed an approach to deal with mismatched domains based on the notion of partial values. A partial value corresponds to a set of possible values in which exactly one is the true value. Tseng et al. [26] extended the concept of partial values to probabilistic partial values, by means of which more informative query results can be provided.

The outerjoin operator [9] is designed to preserve the information of unmatched tuples in the participant relations, which has been included in the SQL2 standard draft [1]. A *one-sided outerjoin* (i.e., *left* or *right outerjoin*) preserves only one of the participant relations while a *two-sided outerjoin* (or *full outerjoin*) preserves both of the participant relations. Many approaches [3, 15, 16, 19, 24] have been proposed to process queries involving outerjoins. Rosenthal and Galindo-Legaria [24] investigated reassociation rules for one-sided outerjoins and presented a special class of join/one-sided outerjoin queries that are freely reorderable. In [15], simplification and reassociation for queries involving one- and two-sided outerjoins were studied, and issues concerning extending traditional optimizers to handle outerjoins were discussed. Strategies for representing outerjoins as order-independent disjunctions and their evaluation were explored in [16]. Pirahesh et al. [23] proposed two specialized algorithms to efficiently process outerjoin queries. Moreover, the algorithms were extended to support parallel execution of the outerjoin operation. Lee and Wiederhold [19] developed a mechanism for prescribing inner joins or left outerjoins for the joins of a query used to instantiate objects from a relational database. In [4], algorithms that remove redundant outer joins from a query were presented. Galindo-Legaria and Rosenthal [17] proposed a theory that allows outerjoin/join queries to be reordered for the sake of optimization. A model of hypergraph abstraction and algorithms for reordering outerjoin queries with complex predicates were proposed by Bhargava et al. [3]. All the above papers considered outerjoin processing in a centralized relational database system.

Since an outerjoin preserves information for the participant relations, it can be used to “union” two semantically related relations/classes in a multidatabase environment. In [18], the integration operator *OUnion* was developed for integrating multiple object databases. The function of the operator is similar to that of the outerjoin operator in the relational model. However, the issue of query processing involving *OUnion* was not further discussed. Dayal [10] used outerjoins to construct a generalized entity type over related entity types from different database systems. To resolve data conflicts, aggregate functions such as “average,” “maximum” etc. were used for the purpose of attribute derivation. However, query optimization involving outerjoins and aggregate functions was not formally discussed.

Chen [7] optimized outerjoin processing by using a set of equivalence transformation rules. A multidatabase query was modified and transformed into one without any outerjoins or one containing only one-sided outerjoins. However, the data conflict problem was not considered. Lim et al. [20] considered the entity identification and attribute value conflict problems in the two-sided outerjoin operation. For the entity identification problem, the *key-equality* comparator was developed to overcome the anomaly caused by the regular equality comparator in two-sided outerjoins. For the attribute conflict problem, the *Generalized Attribute Derivation* (GAD) operation was defined, which can be used to derive new attributes from existing attributes. Moreover, an algebraic transformation framework, including two-sided outerjoins and GAD operations, was proposed for multidatabase queries. In fact, when the data inconsistency problem is not considered, the entity identification problem, as shown in [20], can be resolved by means of natural outerjoins [9]. Furthermore, the GAD operation is similar to the aggregate function presented in [10], which suffers from the problem of losing informative information from component databases.

The natural outerjoin operation is useful in schema integration. A natural outerjoin is an equi-outerjoin on the common attributes with one set of the common attributes preserved in the resultant relation. For example, consider the integration of relation **Student1** (*id*, *name*, *age*, *school*) in a database and relation **Student2** (*id*, *name*, *age*, *address*) in another database. Assume that there is no inconsistent data in these two relations. In addition, their key attributes *id*'s are semantically equivalent; that is, two tuples having the same *id* value are considered to represent the same real-world entity. We can integrate **Student1** and **Student2** to form the view **Student** (*id*, *name*, *age*, *school*, *address*) by the natural outerjoin of **Student1** and **Student2**. If a tuple in **Student1** and another tuple in **Student2** represent the same real-world entity, then a tuple representing this entity, which has both *school* and *address* information from **Student1** and **Student2**, respectively, is obtained when view **Student** is materialized. For a real-world entity which is represented by a tuple in **Student1** but no tuple in **Student2**, the value of the attribute *address* for this entity in **Student** will be filled with a null value denoted as "~" [8].

When the data inconsistency problem is considered, the traditional natural outerjoin cannot be directly used in schema integration. For example, assume there are two tuples (001, John, 25, NTHU) and (001, John, 24, KM100) in **Student1** and **Student2**, respectively. These two tuples represent the same real-world entity since they have the same *id* value, but their *age* values are inconsistent. When view **Student** is materialized, the natural outerjoin of **Student1** and **Student2** is performed. However, these two tuples cannot be joined into a single tuple in the resultant relation due to their inconsistent *age* values. Therefore, there are two tuples (001, John, 25, NTHU, ~) and (001, John, 24, ~, KM100) in **Student**, which will make the user confused about the semantics.

The execution order of left (or right) outerjoins determines the semantics of the result [12]. However, the effect of the execution order on the resultant semantics of natural outerjoins has not been explored before. Because an arbitrary execution order for natural outerjoins may cause the resultant semantics to be ambiguous, care needs to be taken in specifying the execution order. In this paper, the effects of execution orders, inconsistent data and null values on the resultant semantics of natural outerjoins are discussed. We investigate how to determine the execution order of natural outerjoins such that the resultant semantics is *desirable*. Moreover, we find that traditional natural outerjoin cannot be directly applied in defining views when null values and inconsistent data are considered. An extension of the traditional natural outerjoin, called the *partial natural outerjoin,* is thus proposed to handle null values and inconsistent data.

When a user issues a query against a global view, the query is modified to obtain one which may contain partial natural outerjoins, selections, and projections, based on the definition of the global view. A set of equivalence transformation rules are developed to transform a modified query into one with simpler operations, which lowers the query processing cost. Due to the existence of inconsistent data, the selections and projections cannot always be executed at local sites before transmitting relations to a final site where partial natural outerjoins are performed. We discuss cases where the selections and projections can be executed at local sites without affecting the correctness of the query result and provide the corresponding transformation rules for these cases. Moreover, the semijoin technique is applied in query processing. Therefore, the cost of data transmission for processing a query can be further reduced, especially in a wide area network environment.

This paper is organized as follows. In section 2, the effects of execution orders, null values and inconsistent data on the resultant semantics of natural outerjoins are discussed. The partial natural outerjoin is introduced in section 3. Section 4 presents a set of equivalence transformation rules used to optimize queries involving partial natural outerjoins. Finally, we conclude with future work in section 5.

2. SCHEMA INTEGRATION BY MEANS OF NATURAL OUTERJOINS

The operators (*full*) *natural outerjoin*, *left natural outerjoin*, and *right natural outerjoin* are denoted as $\xleftrightarrow{NOJ}_s$, $\xrightarrow{NOJ}_s$, and $\xleftarrow{NOJ}_s$, where "**s**" represents the set of attributes common to the participant relations. Let $R(A1, B1)$ and $S(B1,C1)$ be two relations with attributes $R.A1$, $R.B1$, $S.B1$, and $S.C1$. Attribute $B1$ is the common attribute of these two relations. Define X as the equi-join of R and S on $B1$:

$$X = R \bowtie_{R.B1=S.B1} S.$$

The *natural join* of R and S is defined as follows:

$$X' = \pi_{(R.A_1,R.B_1,S.C_1)} X,$$

where $\pi_{(R.A1,R.B1,S.C1)}$ denotes the projection on attributes $R.A1$, $R.B1$ and $S.C1$. Then we can state the notion of the *antijoin* [10, 15] as follows:

Definition 1: The *antijoin*, denoted as $R \triangleright S$, is defined as $\{t_1 \in R \mid \text{no tuple } t_2 \in S \text{ satisfies } t_1.B_1 = t_2.B_1\}$.

The *natural outerjoin* of R and S is defined as

$$R \xleftrightarrow{NOJ}_{\{B_1\}} S = X' \cup ((R \triangleright S) \times (\sim)_{\{C1\}}) \cup ((\sim)_{\{A1\}} \times (S \triangleright R)),$$

where "$(\sim)_\Phi$" denotes a relation with the attributes in the set Φ, which consists of a null tuple (i.e., a tuple with null values for all the attributes), and "$\times$" represents the Cartesian product. The *left natural outerjoin* and *right natural outerjoin* are defined as follows:

$$R \xrightarrow{NOJ}_{\{B1\}} S = X' \bigcup ((R \triangleright S) \times (\sim)_{\{C1\}}),$$
$$R \xleftarrow{NOJ}_{\{B1\}} S = X' \bigcup ((\sim)_{\{A1\}} \times (S \triangleright R)).$$

The left natural outerjoin preserves information for the left relation of the pair while the right natural outerjoin preserves information for the right relation.

Fig. 1 shows example relations in different databases, where the relations R_1, R_2 and R_3 record information about employees who take part in projects P_1, P_2 and P_3, respectively. Consider the following example.

R_1

$E\#$	$D\#$
E100	D10
E200	D20
E500	D35

R_2

$E\#$	*Ename*	$D\#$
E100	John	D10
E350	Jack	D35
E500	Joe	D35
E800	Jane	D80

R_3

$E\#$	*degree*
E100	BS
E200	BS
E500	PhD
E800	PhD

Fig. 1. Relations in different databases.

Example 1: Suppose we want to create a view V_1(*E#*, *Ename*, *D#*, *degree*) which contains all the information about employees who participate in project P_1, P_2, or P_3. Since the natural outerjoin operation can combine the information for a real-world entity existing in different databases, intuitively, view V_1 can be defined as the natural outerjoins of R_1, R_2, and R_3. (Note that a view is defined after the schema integrator specifies a derivation for it.)

It is obvious that the execution order of left (or right) natural outerjoins determines the semantics of the result. However, the effect of an execution order on the resultant semantics of (full) natural outerjoins has not been explored. In the following, we will investigate the effects of execution orders, null values and inconsistent data on the result of natural outerjoins using the above example.

2.1 The Effect of Execution Orders on the Result of Natural Outerjoins

In this subsection, inconsistent data and null values in base relations of multiple databases are not considered.

Consider **Example 1.** When a schema integrator specifies the natural outerjoins of R_1, R_2 and R_3 as the definition of view V_1, the importance of the execution order for natural outerjoins can be ignored. In fact, different specifications for the execution order will produce different views. Let us consider three possible ways to execute natural outerjoins over R_1, R_2 and R_3: $(R_1 \overset{NOJ}{\longleftrightarrow}_{\{E\#,D\#\}} R_2) \overset{NOJ}{\longleftrightarrow}_{\{E\#\}} R_3$, $R_1 \overset{NOJ}{\longleftrightarrow}_{\{E\#,D\#\}} (R_2 \overset{NOJ}{\longleftrightarrow}_{\{E\#\}} R_3)$, and $R_2 \overset{NOJ}{\longleftrightarrow}_{\{E\#,D\#\}} (R_1 \overset{NOJ}{\longleftrightarrow}_{\{E\#\}} R_3)$. The results are shown in Fig. 2.

From a semantic point of view, we say that a view is *desirable* if a real-world entity is represented by a unique tuple of the view. The result of materializing view V_1 is desirable if it satisfies the following condition: **The employee participating in project *A*, *B*, or *C* is represented by a single tuple in the result**. It can be seen that the result of $(R_1 \overset{NOJ}{\longleftrightarrow}_{\{E\#,D\#\}} R_2) \overset{NOJ}{\longleftrightarrow}_{\{E\#\}} R_3$ is desirable since the information of each employee is integrated into a single tuple. As for the result of $R_1 \overset{NOJ}{\longleftrightarrow}_{\{E\#,D\#\}} (R_2 \overset{NOJ}{\longleftrightarrow}_{\{E\#\}} R_3)$, it is

$(R_1 \overset{NOJ}{\longleftrightarrow}_{\{E\#,D\#\}} R_2) \overset{NOJ}{\longleftrightarrow}_{\{E\#\}} R_3$

E#	*Ename*	*D#*	*degree*
E100	John	D10	BS
E500	Joe	D35	PhD
E200	~	D20	BS
E800	Jane	D80	PhD
E350	Jack	D35	~

$R_1 \overset{NOJ}{\longleftrightarrow}_{\{E\#,D\#\}} (R_2 \overset{NOJ}{\longleftrightarrow}_{\{E\#\}} R_3)$

E#	*Ename*	*D#*	*degree*
E100	John	D10	BS
E500	Joe	D35	PhD
E800	Jane	D80	PhD
E350	Jack	D35	~
E200	~	~	BS
E200	~	D20	~

$R_2 \overset{NOJ}{\longleftrightarrow}_{\{E\#,D\#\}} (R_1 \overset{NOJ}{\longleftrightarrow}_{\{E\#\}} R_3)$

E#	*Ename*	*D#*	*degree*
E100	John	D10	BS
E500	Joe	D35	PhD
E200	~	D20	BS
E800	~	~	PhD
E350	Jack	D35	~
E800	Jane	D80	~

Fig. 2. The results of different execution orders for natural outerjoins over R_1, R_2 and R_3.

undesirable since employee "E200" appears in two tuples in the result. That is, the resultant tuples (E200,~,~,BS) and (E200,~,D20,~) will make the user confused about the semantics. The problem in the execution order is that the tuple (E200,BS) in R_3, which has no match tuple in R_2, is preserved with null *Ename* and *D#* values while the preserved tuple cannot match any tuple in R_1 due to null *D#* value. Similarly, the result of $R_2 \overset{NOJ}{\longleftrightarrow}_{\{E\#,D\#\}}$ $(R_1 \overset{NOJ}{\longleftrightarrow}_{\{E\#\}} R_3)$ is also undesirable.

Theorem 1: Assume that inconsistent data and null values are not considered in the base relations. Given relations R_1, R_2 and R_3, let s_1 be the set of common attributes for R_1 and R_2, R' the result of natural outerjoin over R_1 and R_2, and s_2 the set of common attributes for R' and R_3. Assume R_1, R_2 and R_3 have the same key attribute a_k, and they are to be integrated into a view which contains all the information about entities in R_1, R_2 and R_3. $(R_1 \overset{NOJ}{\longleftrightarrow}_{s_1} R_2) \overset{NOJ}{\longleftrightarrow}_{s_2} R_3$ is a desirable view specification if $s_1 \supseteq s_2$.

Proof: Since null values in base relations are not considered, all the tuples in $R_1 \overset{NOJ}{\longleftrightarrow} s_1 R_2$ have non-null values for each attribute in s_1. Let t be a tuple of R_3, and let t' be a tuple of R'. Consider the following two cases:

- *Case* (a): t and t' represent the same real-world entity.
  Assume $s_1 \supseteq s_2$. Since inconsistent data is not considered, for each attribute a_i in s_2, $t.a_i$ is equal to $t'.a_i$. Therefore, t and t' will be integrated into a single tuple in the result of $R' \overset{NOJ}{\longleftrightarrow}_{s_2} R_3$.
- *Case* (b): t and t' represent different real-world entities.

Since a_k is the key attribute for R_3 and R', $a_k \in s_2$. Moreover, different entities have different key values. Therefore, $t.a_k$ is not equal to $t'.a_k$, and entities represented by t_1 and t_2 will be preserved by two separate tuples in the result of $R' \xleftrightarrow{NOJ}_{s_2} R_3$. Based on cases (a) and (b), we can conclude that $(R_1 \xleftrightarrow{NOJ}_{s_1} R_2) \xleftrightarrow{NOJ}_{s_2} R_3$ is a desirable view specification if $s_1 \supseteq s_2$. □

It is possible that we will not find appropriate sets s_1 and s_2 from given relations such that $s_1 \supseteq s_2$. Consider the following example.

Example 2: Assume relations R_4, R_5 and R_6 in Fig. 3 record the information about employees who participate in projects P_4, P_5 and P_6, respectively. Moreover, we want to create a view V_2(*E#*, *Ename*, *D#*, *degree*, *age*) which contains all the information for employees participating in project P_4, P_5, or P_6.

R_4

E#	*D#*	*degree*
E10	D1	PhD
E80	D3	MS

R_5

E#	*Ename*	*D#*
E20	John	D1
E30	Mary	D2

R_6

E#	*Ename*	*age*
E60	Tom	25
E70	Jane	36

Fig. 3. Relations in different databases.

Let us consider the three possible specifications for defining view V_2: $(R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} R_5) \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6$, $R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} (R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6)$, $R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename,D\#\}} (R_4 \xleftrightarrow{NOJ}_{\{E\#\}} R_6)$. These three specifications have the same result as shown in Fig. 4. It can be seen that the result in Fig. 4 is desirable, though none of the three specifications satisfies the containment condition in **Theorem 1.**

E#	*Ename*	*D#*	*degree*	*age*
E10	~	D1	PhD	~
E20	John	D1	~	~
E30	Mary	D2	~	~
E60	Tom	~	~	25
E70	Jane	~	~	26
E80	~	D3	MS	~

Fig. 4. The same result for $(R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} R_5) \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6$ and $R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} (R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6)$, and $R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename,D\#\}} (R_4 \xleftrightarrow{NOJ}_{\{E\#\}} R_6)$.

Suppose the tuples (E10,Peter,D1) and (E80,Jack,20) are inserted into relations R_5 and R_6, respectively. The results of the three specifications will change to those shown in Fig. 5. Note that $(R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} R_5) \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6$ and $R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} (R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6)$ have the same result; however, the result is undesirable since employee "E80" is represented by two tuples. Similarly, the result of $R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename,D\#\}} (R_4 \xleftrightarrow{NOJ}_{\{E\#\}} R_6)$ is also undesirable since employee "E10" is represented by two tuples.

From the above discussion, it can be seen that if the condition $s_1 \supseteq s_2$ in Theorem 1 does not apply, whether or not the result of $(R_1 \xleftrightarrow{NOJ}_{s_1} R_2) \xleftrightarrow{NOJ}_{s_2} R_3$ is desirable depends on the data in relations R_1, R_2 and R_3.

$(R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} R_5) \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6$

$R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} (R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6)$

E#	*Ename*	*D#*	*degree*	*age*
E10	Peter	D1	PhD	~
E20	John	D1	~	~
E30	Mary	D2	~	~
E60	Tom	~	~	25
E70	Jane	~	~	36
E80	~	D3	MS	~
E80	Jack	~	~	20

$R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename,D\#\}} (R_4 \xleftrightarrow{NOJ}_{\{E\#\}} R_6)$

E#	*Ename*	*D#*	*degree*	*age*
E10	~	D1	PhD	~
E10	Peter	D1	~	~
E20	John	D1	~	~
E30	Mary	D2	~	~
E60	Tom	~	~	25
E70	Jane	~	~	36
E80	Jack	D3	MS	20

Fig. 5. The results for $(R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} R_5) \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6$, $R_4 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} (R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename\}} R_6)$, and $R_5 \xleftrightarrow{NOJ}_{\{E\#,Ename,D\#\}} (R_4 \xleftrightarrow{NOJ}_{\{E\#\}} R_6)$.

2.2 The Effect of Null Values and Inconsistent Data on the Result of Natural Outerjoins

In this subsection, inconsistent data and null values in base relations of multiple databases are considered. We will find that even if the execution order of natural outerjoins satisfies the containment condition in **Theorem 1**, null values and inconsistent data may cause the result to be ambiguous.

We replace relation R_2 with R'_2 as shown in Fig. 6 for the following discussion. Consider **Example 1** again. The result of $(R_1 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} R_2') \xleftrightarrow{NOJ}_{\{E\#\}} R_3$ is depicted in Fig. 7. It can be seen that the specification $(R_1 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} R_2') \xleftrightarrow{NOJ}_{\{E\#\}} R_3$ satisfies the containment condition in **Theorem 1**; however, the execution result is ambiguous since employees "E100" and "E500" are both represented by two tuples in the result. This is because null values and inconsistent data prevent the tuples representing the same employees from matching in natural outerjoin operations. For example, observe the tuple (E100, D10) in R_1 and the tuple (E100,John,~) in R'_2. These two tuples represent the same employee, but the *D#* value in R'_2 is null. As a result, the condition $R_1.D\# = R'_2.D\#$, which is implied in the operation $R_1 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} R_2'$, is not satisfied. Similarly, the two tuples (E500,D35) in R_1 and (E500,Joe,D20) in R'_2 do not satisfy the condition $R_1.D\# = R'_2.D\#$ due to their inconsistent *D#* values.

E#	*Ename*	*D#*
E100	John	~
E350	Jack	D35
E500	Joe	D20
E800	Jane	D80

Fig. 6. Relation R'_2.

E#	*Ename*	*D#*	*degree*
E100	~	D10	BS
E200	~	D20	BS
E500	~	D35	PhD
E100	John	~	BS
E500	Joe	D20	PhD
E800	Jane	D80	PhD
E350	Jack	D35	~

Fig. 7. The result of $(R_1 \xleftrightarrow{NOJ}_{\{E\#,D\#\}} R_2') \xleftrightarrow{NOJ}_{\{E\#\}} R_3$.

Null values and inconsistent data also have similar effects on the results of left natural outerjoins and right natural outerjoins. In the next section, we will develop a new outerjoin operator, called the *partial natural outerjoin*, to handle problems caused by execution orders, null values, and inconsistent data.

3. PARTIAL NATURAL OUTERJOIN

In section 3.1, we will formally define the partial natural outerjoin. Examples for illustrating the applications and advantages of the partial natural outerjoin are given in section 3.2.

3.1 Definition

First we will introduce the *data integration* operator, denoted as $\uplus$, which is used in the definition of the partial natural outerjoin. The probabilistic technique and our previous research on *probabilistic partial values* [26] will be used to deal with value conflicts in data integration.

Definition 2: A *probabilistic partial value*, denoted $\left[u_1^{x_1}, u_2^{x_2}, \cdots, u_m^{x_m}\right]$, is a set of possible values with a probability assigned to each possible value, in which exactly one possible value is the true value, where $\{u_1, u_2, ..., u_m\}$ is the set of possible values, x_i is the associated probability of u_i and $\sum_{i=1}^{m} x_i = 1$.

Note that a definite value, say v, can be expressed as a probabilistic partial value $[v^1]$. Two values are equal only when they have the same definite data value. The data integration operator, denoted as $\uplus$, is defined as follows.

Definition 3: Let a and b be two values to be integrated. $a \uplus b$ is defined as follows:

- **Case 1:** a and b are equal
  $a \uplus b \equiv a$.
- **Case 2:** a and b are unequal
  1. a or b is a null value:
     $a \uplus \sim \equiv a$,
     $b \uplus \sim \equiv b$,
     $\sim \uplus \sim \equiv \sim$.
  2. a and b are both non-null values:
     Assume a and b are $\left[u_1^{x_1}, u_2^{x_2}, \cdots, u_m^{x_m}\right]$ and $\left[v_1^{y_1}, v_2^{y_2}, \cdots, v_n^{y_n}\right]$, respectively. Let r_1 and r_2 be the *reliabilities* of $\left[u_1^{x_1}, u_2^{x_2}, \cdots, u_m^{x_m}\right]$ and $\left[v_1^{y_1}, v_2^{y_2}, \cdots, v_n^{y_n}\right]$, respectively, and $r_1 + r_2 = 1$:

$$a \uplus b \equiv$$
$$\left\{w^k \mid (\exists u_i^{x_i})(\exists \upsilon_j^{y_j})(u_i^{x_i} \in a \wedge \upsilon_j^{y_i} \in b \wedge \omega = u_i = \upsilon_j \wedge k = (r_1 \times x_i + r_2 \times y_j))\right\} \cup$$
$$\left\{w^k \mid (\exists u_i^{x_i})(u_i^{x_i} \in a \wedge w = u_i \wedge (\nexists \upsilon_j^{y_i}) = (\upsilon_j^{y_i} \in b \wedge u_i = \upsilon_j) \wedge k = r_1 \times x_i)\right\} \cup$$
$$\left\{w^k \mid (\exists \upsilon_j^{y_j})(\upsilon_j^{y_j} \in b \wedge w = \upsilon_j \wedge (\nexists u_i^{x_i})(u_i^{x_i} \in a \wedge \upsilon_j = u_i) \wedge k = r_2 \times y_j)\right\}.$$

Note that if both a and b are definite values, then $a \uplus b$ is $[a^{r_1}, b^{r_2}]$. For the sake of simplicity, we assume $r_1 = r_2 = \frac{1}{2}$ in the following discussion.

The data integration operator is not the only operator which can be used to resolve value conflicts. However, we believe that a notation with quantitative probabilities is more informative than one with just a single value [10, 20].

Let $\xleftrightarrow{PNO}_S$, $\xrightarrow{PNO}_S$, and $\xleftarrow{PNO}_s$ represent the operators (*full*) *partial natural outerjoin*, *left partial natural outerjoin*, and *right partial natural outerjoin*, respectively, where the symbol "*S*" denotes a subset of the common attributes of the participant relations. Because tuples from different databases, which represent the same real-world entity, may have different values for their common attributes, the traditional natural outerjoin may fail to join these tuples to form a single tuple as illustrated in section 2.2. The partial natural outerjoin is developed to allow the user to explicitly specify partial common attributes as the outerjoin attributes represented by *S*. We call the attribute in *S* an *identifying attribute*. In the partial natural outerjoin operation, two tuples with the same value for each identifying attribute are considered to represent the same real-world entity and are joined. In other words, we assume that if an attribute is specified as an identifying attribute, then there is no data inconsistency in the values of the attribute. As for those common attributes not in *S*, they may have inconsistent data in the participant relations, and the data integration operator can be used to integrate inconsistent data using the technique of probabilistic partial values.

The partial natural outerjoin is defined as follows. An attribute *attr* is called a *private attribute* for relation R_i, where i = 1, 2, if *attr* appears in R_i and is not common to any attribute of the other relation. For relations R_1 and R_2, assume

S is the set of identifying attributes a_1, a_2, ..., and a_k;
$S_{c'}$ is the set of common but not identifying attributes c_1, c_2, ..., and c_j;
S_{p1} is the set of private attributes x_1, x_2, ..., and x_u for R_1;
S_{p2} is the set of private attributes y_1, y_2, ..., and y_v for R_2.

The partial natural outerjoin of R_1 and R_2 produces a relation with scheme (S, $S_{c'}$, S_{p1}, S_{p2}). Let t_1 and t_2 be two tuples in R_1 and R_2, respectively. The function $P_S(t_1, t_2)$ is defined as

$$P_s(t_1,t_2) = \begin{cases} \mathit{true} & \text{if } t_1 \cdot a_i = t_2 \cdot a_i \text{ for each attribute } a_i \text{ in } S. \\ \mathit{false} & \text{otherwise} \end{cases}$$

Definition 4: The partial natural join of R_1 and R_2, denoted as $R_1 \overset{PNJ}{\bowtie}_S R_2$, is defined as

$$\begin{aligned} R_1 \overset{PNJ}{\bowtie}_S R_2 \\ & \{t \mid \exists t_1)(\exists t_2)(t_1 \in R_1 \wedge t_2 \in R_2 \wedge Ps(t_1,t_2) = \text{"}\mathit{true}\text{"} \wedge \\ & (t \cdot a_i = t_1 \cdot a_i, a_i \in S) \wedge \\ & (t \cdot c_i = t_1 \cdot c_i \uplus t_2 \cdot c_i, c_i \in S_{c'}) \wedge \\ & (t \cdot x_i = t_1 \cdot x_i, x_i \in S_{p1}) \wedge \\ & (t \cdot y_i = t_2 \cdot y_i, y_i \in S_{p2}))\} \end{aligned}$$

$R_1 \overset{PNJ}{\bowtie}_S R_2$ represents the result of integrating the tuples in R_1 and R_2, which have the same value for each identifying attribute.

Let $\overline{R_1}$ and $\overline{R_2}$ be the sets of *extended unmatched tuples* in R_1 and R_2, respectively, with respect to the partial natural join of R_1 and R_2:

$$\begin{aligned}
&\overline{R_1} \equiv \{t|\exists t_1)(t_1 \in R_1 \wedge (\nexists t_2)(t_2 \in R_2 \wedge P_s(t_1,t_2) = \text{“}true\text{”}) \wedge \\
&(t \cdot a_i = t_1 \cdot a_i,\ a_i \in S) \wedge \\
&(t \cdot c_i = t_1 \cdot c_i,\ c_i \in S_{c'}) \wedge \\
&(t \cdot x_i = t_1 \cdot x_i,\ x_i \in S_{p1}) \wedge \\
&(t \cdot y_i = \text{“}\sim\text{”},\ y_i \in S_{p2}))\}
\end{aligned}$$

and

$$\begin{aligned}
&\overline{R_2} \equiv \{t|\exists t_2)(t_2 \in R_2 \wedge (\nexists t_1)(t_1 \in R_1 \wedge P_s(t_1,t_2) = \text{“}true\text{”}) \wedge \\
&(t \cdot a_i = t_2 \cdot a_i,\ a_i \in S) \wedge \\
&(t \cdot c_i = t_2 \cdot c_i,\ c_i \in S_{c'}) \wedge \\
&(t \cdot x_i = \text{“}\sim\text{”},\ x_i \in S_{p1}) \wedge \\
&(t \cdot y_i = t_2 \cdot y_i,\ y_i \in S_{p2}))\}.
\end{aligned}$$

Definition 5: The partial natural outerjoin of R_1 and R_2 is defined as

$$R_1 \xleftrightarrow{PNO}_s R_2 = R_1 \overset{PNJ}{\bowtie} {}_s R_2 \cup \overline{R_1} \cup \overline{R_2}\ .$$

The left partial natural outerjoin and right partial natural outerjoin are defined as

$$R_1 \xleftarrow{PNO}_s R_2 = R_1 \overset{PNJ}{\bowtie} {}_s R_2 \cup R_2 \cup \overline{R_1}\ .$$

and

$$R_1 \xrightarrow{PNO}_s R_2 = R_1 \overset{PNJ}{\bowtie} {}_s R_2 \cup R_2 \cup \overline{R_2}\ ,$$

respectively.

The result of performing partial natural outerjoins may contain probabilistic partial values. In [26], we developed a set of extended relational operators for manipulating relations containing probabilistic partial values. The partial natural outerjoin operators and the previously proposed extended relational operators can be combined to support a more powerful set of operations for use in a multidatabase system.

3.2 Examples

The partial natural outerjoin is useful for the schema integrator to define desirable views. Consider Example 1 again. We replace relation R_2 with R'_2 as shown in Fig. 6 for the purpose of discussion. View V_1 can be defined as $(R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2') \xleftrightarrow{PNO}_{\{E\#\}} R_3$ whose result is shown in Fig. 8. It can be seen that the result shown in Fig. 8 is more desirable than that shown in Fig. 7. In other words, the problems caused by inconsistent data and null values have been resolved by partial natural outerjoins.

E#	*Ename*	*D#*	*degree*
E100	John	D10	BS
E500	Joe	$[D35^{\frac{1}{2}}, D20^{\frac{1}{2}}]$	PhD
E200	~	D20	BS
E800	Jane	D80	PhD
E350	Jack	D35	~

Fig. 8. The result of $(R_1 \overset{PNO}{\longleftrightarrow}_{\{E\#\}} R_2') \overset{PNO}{\longleftrightarrow}_{\{E\#\}} R_3$.

Consider the following case for **Example 2**. Assume the tuples (E10,Peter, D1) and (E80, Jack, 20) are inserted into relations R_5 and R_6 shown in Fig. 3, respectively. View V_2 can be defined as $(R_4 \overset{PNO}{\longleftrightarrow}_{\{E\#\}} R_5) \overset{PNO}{\longleftrightarrow}_{\{E\#\}} R_6$, whose result is shown in Fig. 9. It can be seen that the result shown in Fig. 9 is more desirable than that shown in Fig. 5 since each employee is represented by a single tuple in Fig. 9. In section 2.1, we found that for **Example 2**, whether or not the result of natural outerjoins is desirable is data dependent. Now by means of partial natural outerjoins, the problem caused by data dependency can be resolved.

E#	*Ename*	*D#*	*degree*	*age*
E10	Peter	D1	PhD	~
E20	John	D1	~	~
E30	Mary	D2	~	~
E60	Tom	~	~	25
E70	Jane	~	~	36
E80	Jack	D3	MS	20

Fig. 9. The result of $(R_4 \overset{PNO}{\longleftrightarrow}_{\{E\#\}} R_5) \overset{PNO}{\longleftrightarrow}_{\{E\#\}} R_6$.

Consider another example. Assume that relation **Teacher**(*id*, *name*, *specialty*, *age*) in one database records the data of teachers at X University, and that relation **Consultant** (*id*, *name*, *specialty*, *age*, *degree*) in another database records the data of consultants at Y Company. The key attributes *id* in **Teacher** represent the identification number for teachers while that in **Consultant** represents the identification number for consultants. Obviously, the keys in **Teacher** and **Consultant** are incompatible. If the schema integrator wants to build a view called **Teacher_Consultant**, which represents the persons who teach at X University and consult at Y Company, then the identifying attributes can be used to identify the same persons. Assume attributes *name* and *specialty* are identifying attributes. The schema integrator can define view **Teacher_Consultant** as **Teacher** $\overset{PNJ}{\bowtie}_{\{name,\ specialty\}}$ **Consultant**. That is, if two tuples from **Teacher** and **Consultant** have the same values for attributes *name* and *specialty*, then they are considered to represent the same person. The handling of incompatible keys was described in [25].

4. OPTIMIZING PARTIAL NATURAL OUTERJOINS BY MEANS OF ALGEBRAIC QUERY TRANSFORMATION

Assume the execution order of partial natural outerjoins has been determined by the schema integrator. In query processing, a global query is first modified to obtain a query which only refers to local relations based on the definition of the global view. Then, the modified query is decomposed into subqueries to be executed in local databases. To enhance readability, we will relegate basic rules (**B1**), (**B2**), ..., and (**B13**) to Appendix A.

Based on these basic rules, we propose a set of equivalence transformation rules for optimizing queries involving partial natural outerjoins in sections 4.1, 4.2 and 4.3. The notations used in the following discussion are described in Table 1. We assume relations R_1, R_2, and R_3 reside at different sites.

Table 1. The notations.

notation	*description*
R_i	a local relation in a database, $i = 1, 2, 3$
A_i	the set of attributes in R_i, $i = 1, 2, 3$
S	the set of identifying attributes for R_1 and R_2
$S_{c'}$	the set of common but not identifying attributes for R_1 and R_2
P_i	the set of *private attributes* for R_i, $i = 1, 2$
R'	$R' = R_1 \overset{PNO}{\longleftrightarrow}_S R_2$
B	the set of identifying attributes for R' and R_3

Let p be a selection predicate of the form "*attr* **op** *C*." *attr* denotes an attribute, **op** is an operator, such as ">," "<" or "=," and C is a constant. The *associated attribute* of p is defined as the "*attr*" component. Let a_i be the associated attribute of p. Note that $\overline{R_1}$ and $\overline{R_2}$ represent the sets of extended unmatched tuples in R_1 and R_2 with respect to the operation $R_1 \overset{PNJ}{\bowtie}_S R_2$, respectively, as defined in section 3.1.

Based on the fact that the execution cost of $R_1 \overset{PNJ}{\bowtie}_S R_2$ is less than that of $R_1 \xrightarrow{PNO}_S R_2$ and that of $R_1 \xleftarrow{PNO}_S R_2$, that the execution costs of $R_1 \xrightarrow{PNO}_S R_2$ and $R_1 \xleftarrow{PNO}_S R_2$ are less than that of $R_1 \overset{PNO}{\longleftrightarrow}_S R_2$, and that the execution cost of $R_2 \xrightarrow{SJ} R_1$ is less than that of $R_1 \overset{PNJ}{\bowtie}_S R_2$, we will discuss the algebraic query transformation rules in the following section for query optimization. Note that $R_2 \xrightarrow{SJ} R_1$ represents a semijoin from R_2 to R_1 on S.

4.1 Simplification of Partial Natural Outerjoins Followed by Selections

Assume the operators have the following precedence order: selection, partial natural (outer)join, and union. Two cases will be considered for simplifying partial natural outerjoins followed by selections.

case 1: $a_i \in P_1$

- **Rule (1.1):** $\sigma_p(R_1 \xrightarrow{PNO}_S R_2) = (\sigma_p R_1) \xrightarrow{PNO}_S R_2$

  Proof.

$$\begin{aligned}
\sigma_p(R_1 \overset{PNO}{\longleftrightarrow}_S R_2) &= \sigma_p(R_1 \overset{PNJ}{\bowtie}_S R_2 \cup \overline{R_1} \cup \overline{R_2}), \text{by definition} \\
&= \sigma_p(R_1 \overset{PNJ}{\bowtie}_S R_2) \cup (\sigma_p \overline{R_1}) \cup (\sigma_p \overline{R_2}), \text{ by distributive law} \\
&= \sigma_p(R_1 \overset{PNJ}{\bowtie}_S R_2) \cup (\sigma_p \overline{R_1}), \text{ by } \textbf{(B1)} \\
&= (\sigma_p R_1 \overset{PNJ}{\bowtie}_S R_2) \cup (\sigma_p \overline{R_1}), \text{ by } \textbf{(B2)} \\
&= (\sigma_p R_1 \overset{PNJ}{\bowtie}_S R_2) \cup (\overline{\sigma_p R_1}), \text{ by } \textbf{(B3)} \\
&= (\sigma_p R_1) \xrightarrow{PNO}_S R_2, \text{ by definition}
\end{aligned}$$

□

- **Rule (1.2):** $\sigma_p(R_1 \xrightarrow{PNO}_S R_2) = (\sigma_p R_1) \xrightarrow{PNO}_S R_2$

Proof. The proof is similar to thhat of **Rule (1.1)**. ☐

- **Rule (1.3):** $\sigma_p(R_1 \xleftarrow{PNO}_S R_2) = (\sigma_p R_1) \overset{PNJ}{\bowtie}_S R_2$

Proof. The proof is similar to that of **Rule (1.1)**. ☐

For the case where $a_i \in P_2$, similar rules can be derived.

case 2: $a_i \in S$

- **Rule (2.1):** $\sigma_p(R_1 \xleftrightarrow{PNO}_S R_2) = (\sigma_p R_1) \xleftrightarrow{PNO}_S (\sigma_p R_2)$

Proof.

$$\sigma_p(R_1 \xleftrightarrow{PNO}_S R_2) = \sigma_p(R_1 \overset{PNJ}{\bowtie}_S R_2 \cup \overline{R_1} \cup \overline{R_2}), \text{by definition}$$
$$= \sigma_p(R_1 \overset{PNJ}{\bowtie}_S R_2) \cup (\sigma_p \overline{R_1}) \cup (\sigma_p \overline{R_2}), \text{ by distributive law}$$
$$= (\sigma_p R_1) \overset{PNJ}{\bowtie}_S (\sigma_p R_2) \cup (\sigma_p \overline{R_1}) \cup (\sigma_p \overline{R_2}), \text{by } \textbf{(B4)}$$
$$= (\sigma_p R_1) \overset{PNJ}{\bowtie}_S (\sigma_p R_2) \cup (\overline{\sigma_p R_1}) \cup (\overline{\sigma_p R_2}), \text{by } \textbf{(B5) (B6)}$$
$$= (\sigma_p R_1) \xleftrightarrow{PNO}_S (\sigma_p R_2), \text{ by definition}$$

☐

- **Rule (2.2):** $\sigma_p(R_1 \xrightarrow{PNO}_S R_2) = (\sigma_p R_1) \xrightarrow{PNO}_S (\sigma_p R_2)$

Proof. The proof is similar to that of **Rule (2.1).** ☐

- **Rule (2.2):** $\sigma_p(R_1 \xleftarrow{PNO}_S R_2) = (\sigma_p R_1) \xleftarrow{PNO}_S (\sigma_p R_2)$

Proof. The proof is similar to that of **Rule (2.1).** ☐

4. 2 Simplification of Partial Natural Outerjoins Followed by Projections

In the following, three cases will be considered for simplifying partial natural outerjoins followed by projections. Let Q be the set of attributes to be projected. Assume that null tuples are discarded, and that duplicate tuples are not allowed in a relation.

case 1: $Q \subseteq P_1$

- **Rule (3.1):** $\pi_Q(R_1 \xleftrightarrow{PNO}_S R_2) = \pi_Q R_1$

Proof.

$$\pi_Q(R_1 \xleftrightarrow{PNO}_S R_2) = \pi_Q(R_1 \overset{PNJ}{\bowtie}_S R_2 \cup \overline{R_1} \cup \overline{R_2}), \text{ by definition}$$
$$= \pi_Q(R_1 \overset{PNJ}{\bowtie}_S R_2) \cup \pi_Q(\overline{R_1}) \cup \pi_Q(\overline{R_2}), \text{ by distributive law}$$
$$= \pi_Q(R_1 \overset{PNJ}{\bowtie}_S R_2) \cup \pi_Q(\overline{R_1}), \text{ by } \textbf{(B9)}$$
$$= \pi_Q \hat{R}_1 \cup \pi_Q \tilde{R}_1, \text{by } \textbf{(B7) (B8)}$$
$$= \pi_Q(\hat{R}_1 \cup \tilde{R}_1)$$
$$= \pi_Q R_1, \text{ by definition}$$

- **Rule (3.2):** $\pi_Q(R_1 \xrightarrow{PNO}_S R_2) = \pi_Q R_1$
  Proof. The proof is similar to that of **Rule (3.1).** □

- **Rule (3.3):** $\pi_Q(R_1 \xleftarrow{PNO}_S R_2) = \pi_Q(R_2 \xrightarrow{SJ}_S R_1)$
  Proof. The proof is similar to that of **Rule (3.1).** □

For the case where $Q \subseteq P_2$, similar rules can be derived.

case 1: $Q \subseteq S$

- **Rule (4.1):** $\pi_Q(R_1 \xleftrightarrow{PNO}_S R_2) = (\pi_Q R_1) \cup (\pi_Q R_2)$
  Proof.

$$\pi_Q(R_1 \xleftrightarrow{PNO}_S R_2) = \pi_Q(R_1 \overset{PNJ}{\bowtie}_S R_2 \cup \overline{R_1} \cup \overline{R_2}), \text{ by definition}$$
$$= \pi_Q(R_1 \overset{PNJ}{\bowtie}_S R_2) \cup (\pi_Q \overline{R_1}) \cup (\pi_Q \overline{R_2}), \text{ by distributive law}$$
$$= (\pi_Q \hat{R}_1) \cup (\pi_Q \hat{R}_2) \cup (\pi_Q \overline{R_1}) \cup (\pi_Q \overline{R_2}), \text{by } \textbf{(B10)}$$
$$(\text{Note: } \pi_Q \hat{R}_1 = \pi_Q \hat{R}_2 = (\pi_Q \hat{R}_1) \cup (\pi_Q \hat{R}_2))$$
$$= (\pi_Q \hat{R}_1) \cup (\pi_Q \hat{R}_2) \cup (\pi_Q \tilde{R}_1) \cup (\pi_Q \tilde{R}_2), \text{ by } \textbf{(B11) (B12)}$$
$$= (\pi_Q R_1) \cup (\pi_Q R_2), \text{ by definition}$$

- **Rule (4.2):** $\pi_Q(R_1 \xrightarrow{PNO}_S R_2) = \pi_Q R_1$
  Proof. The proof is similar to that of **Rule (4.1).** □

- **Rule (4.3):** $\pi_Q(R_1 \xleftarrow{PNO}_S R_2) = \pi_Q R_2$
  Proof. The proof is similar to that of **Rule (4.1).** □

case 3: $S_1 \subseteq P_1$, $S_2 \subseteq S$, and $Q = S_1 \cup S_2$

- **Rule (5.1):** $\pi_Q(R_1 \xrightarrow{PNO}_S R_2) = \pi_Q R_1$
  Proof. The rule can be derived from **Rule (3.2)** and **Rule (4.2).** □

For the case where $S_1 \subseteq P_2$, $S_2 \subseteq S$, and $Q = S_1 \cup S_2$, the corresponding rule can be derived.

4.3 Simplification of Multiple Partial Natural Outerjoins

In the following, cases for simplifying multiple partial natural outerjoins will be considered.

case 1: $B \cap P_2 \neq \phi$

- **Rule (6.1):** $(R_1 \xleftrightarrow{PNO}_S R_2) \xleftarrow{PNO}_B R_3 = (R_1 \xleftarrow{PNO}_S R_2) \xleftarrow{PNO}_B R_3$
  Proof.

$$
\begin{aligned}
(R_1 \xleftrightarrow{PNO}_S R_2) \xleftarrow{PNO}_B R_3 &= (R_1 \overset{PNJ}{\bowtie}_S R_2 \cup \overline{R_1} \cup \overline{R_2}) \xleftarrow{PNO}_B R_3, \text{ by definition} \\
&= ((R_1 \overset{PNJ}{\bowtie}_S R_2 \cup \overline{R_1} \cup \overline{R_2}) \overset{PNJ}{\bowtie}_B R_3) \cup \overline{R_3}, \text{ by definition} \\
&= ((R_1 \overset{PNJ}{\bowtie}_S R_2) \overset{PNJ}{\bowtie}_B R_3) \cup (\overline{R_1} \overset{PNJ}{\bowtie}_B R_3) \cup (\overline{R_2} \overset{PNJ}{\bowtie}_B R_3) \cup \overline{R_3}, \\
&\quad \text{by distributive law} \\
&= ((R_1 \overset{PNJ}{\bowtie}_S R_2) \overset{PNJ}{\bowtie}_B R_3) \cup (\overline{R_2} \overset{PNJ}{\bowtie}_B R_3) \cup \overline{R_3}, \text{ by } \mathbf{(B13)} \\
&= ((R_1 \overset{PNJ}{\bowtie}_S R_2 \cup \overline{R_2}) \overset{PNJ}{\bowtie}_B R_3) \cup \overline{R_3} \\
&= (R_1 \xleftarrow{PNO}_S R_2) \overset{PNJ}{\bowtie}_B R_3 \cup \overline{R_3}, \text{ by definition} \\
&= (R_1 \xleftarrow{PNO}_S R_2) \overset{PNO}{\bowtie}_B R_3, \text{ by definition}
\end{aligned}
$$
□

- **Rule (6.2):** $(R_1 \xrightarrow{PNO}_S R_2) \xleftarrow{PNO}_B R_3 = (R_1 \overset{PNJ}{\bowtie}_S R_2) \xleftarrow{PNO}_B R_3$
  Proof. The proof is similar to that of **Rule (6.1)** □

- **Rule (6.3):** $(R_1 \xleftrightarrow{PNO}_S R_2) \overset{PNJ}{\bowtie}_B R_3 = (R_1 \xleftarrow{PNO}_S R_2) \overset{PNJ}{\bowtie}_B R_3$
  Proof. The proof is similar to that of **Rule (6.1)** □

- **Rule (6.4):** $(R_1 \xrightarrow{PNO}_S R_2) \overset{PNJ}{\bowtie}_B R_3 = (R_1 \overset{PNJ}{\bowtie}_S R_2) \overset{PNJ}{\bowtie}_B R_3$
  Proof. The proof is similar to that of **Rule (6.1)** □

For the case where $B \cap P_1 \neq \phi$, similar rules can be derived.

4.4 An Example

The rules proposed in this paper can be used to simplify query processing using the procedure shown in Fig. 10. Consider the following relations:

Procedure P
Step 1: If the query involves multiple partial natural outerjoins, simplify the query by rules in section 4.3.
Step 2: If the query involves selections, simplify the query by rules in section 4.1, and basic rules (**B2**) and (**B4**).
Step 3: If the query involves projections, simplify the query by rules in section 4.2, and basic rules (**B7**) and (**B10**).

Fig. 10. The procedure for simplifying queries involving partial natural outerjoins.

R_1(*E#*, *project*, *degree*),
R_2(*E#*, *Ename*, *project*),
R_3(*Ename*, *address*, *age*),
R_4(*project*, *manager*).

Assume the schema integrator defines a view V as

$((R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2) \xleftarrow{PNO}_{\{Ename\}} R_3) \overset{PNJ}{\bowtie}_{\{project\}} R_4$, Consider a query $\sigma_{\text{degree=PhD}}(V)$. The query is processed by procedure ***P*** as follows:

$$
\begin{aligned}
&\sigma_{degree=PhD}(V)\\
&= \sigma_{degree=PhD}(((R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2) \xleftarrow{PNO}_{\{Ename\}} R_3) \overset{PNJ}{\bowtie}_{\{project\}} R_4), \text{ by view definition}\\
&= \sigma_{degree=PhD}((R_3 \xrightarrow{PNO}_{\{Ename\}} (R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2)) \overset{PNJ}{\bowtie}_{\{project\}} R_4), \text{ by rearrangement}\\
&= \sigma_{degree=PhD}((R_3 \overset{PNJ}{\bowtie}_{\{Ename\}} (R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2)) \overset{PNJ}{\bowtie}_{\{project\}} R_4), \text{ by rule } \textbf{(6.4)}\\
&= \sigma_{degree=PhD}(((R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2) \overset{PNJ}{\bowtie}_{\{Ename\}} R_3) \overset{PNJ}{\bowtie}_{\{project\}} R_4), \text{ by rearrangement}\\
&= \sigma_{degree=PhD}(((R_1 \xleftarrow{PNO}_{\{E\#\}} R_2) \overset{PNJ}{\bowtie}_{\{Ename\}} R_3) \overset{PNJ}{\bowtie}_{\{project\}} R_4), \text{by rule } \textbf{(6.3)}\\
&= ((\sigma_{degree=PhD}((R_1 \xleftarrow{PNO}_{\{E\#\}} R_2) \overset{PNJ}{\bowtie}_{\{Ename\}} R_3)) \overset{PNJ}{\bowtie}_{\{project\}} R_4), \text{by basic rule } \textbf{(B2)}\\
&= (((\sigma_{degree=PhD}((R_1 \xleftarrow{PNO}_{\{E\#\}} R_2)) \overset{PNJ}{\bowtie}_{\{Ename\}} R_3) \overset{PNJ}{\bowtie}_{\{project\}} R_4), \text{by basic rule } \textbf{(B2)}\\
&= ((((\sigma_{degree=PhD} R_1) \overset{PNJ}{\bowtie}_{\{E\#\}} R_2) \overset{PNJ}{\bowtie}_{\{Ename\}} R_3) \overset{PNJ}{\bowtie}_{\{project\}} R_4), \text{by rule } \textbf{(1.3).}
\end{aligned}
$$

The resultant query is one with a lower processing cost.

5. DISCUSSION AND FUTURE WORK

The contributions of this paper are summarized in the following:

1. The effect of the execution order on the resultant semantics of traditional natural outerjoins has been discussed. **Theorem 1** has been proposed to help the user specify the execution order of natural outerjoins in order to obtain a desirable result.
2. We find that the traditional natural outerjoin cannot be directly applied to schema integration when null values and inconsistent data are considered in a multidatabase environment. An extension of the traditional natural outerjoin, called the *partial natural outerjoin*, has been proposed to handle this case. In the partial natural outerjoin operation, we use probabilistic partial values to resolve value conflicts. Of course, this is not the only possible method for resolving this problem. However, we believe that a notation with quantitative probabilities is more informative than one with only a single value based on traditional aggregate functions [10, 20]. Moreover, for non-numerical value conflicts, traditional aggregate functions, such as "average," "maximum" etc., are unable to deal with this case while the approach with probabilistic partial values still works well.
3. A set of equivalence transformation rules has been developed for optimizing queries involving partial natural outerjoins. The transformation rules transform a query into one with simpler operations, which lowers the query processing cost. We have also discussed the cases in which the selection and projection operations can be executed at

local sites without affecting the correctness of the query result, and we have provided the corresponding transformation rules for these cases. Moreover, the semijoin technique has been applied in query processing. Therefore, the cost of data transmission for processing a query can be reduced, especially in a wide area network environment.

In addition to simplification, the reordering of operations is considered to be an important technique for query optimization. However, reordering is difficult to use in approaches that use probabilistic partial values or aggregate functions to resolve value conflicts in a multidatabase environment. Consider the relations shown in Fig. 11 and the following two cases:

R_1

$E\#$	*salary*
E10	120

R_2

$E\#$	$D\#$	*salary*
E10	D20	100

R_3

$E\#$	*degree*	*salary*
E10	MS	100

Fig. 11. Example relations.

- **Case 1:** Partial natural outerjoins $(R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2) \xleftrightarrow{PNO}_{\{E\#\}} R_3$ are used to combine the information for employee "E10" in R_1, R_2 and R_3.
  $(R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2) \xleftrightarrow{PNO}_{\{E\#\}} R_3$ will produce a single tuple for "E10." Another execution order $R_1 \xleftrightarrow{PNO}_{\{E\#\}} (R_2 \xleftrightarrow{PNO}_{\{E\#\}} R_3)$ produces a single tuple for "E10" as well. However, $(R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2) \xleftrightarrow{PNO}_{\{E\#\}} R_3$ is not equivalent to $R_1 \xleftrightarrow{PNO}_{\{E\#\}} (R_2 \xleftrightarrow{PNO}_{\{E\#\}} R_3)$ as shown in Fig. 12. This is because the data integration operator is not associative.

$(R_1 \overset{PNO}{\longleftrightarrow}_{\{E\#\}} R_2) \overset{PNO}{\longleftrightarrow}_{\{E\#\}} R_3$

$E\#$	$D\#$	*degree*	salary
E10	D20	MS	$[120^{\frac{1}{4}}, 100^{\frac{3}{4}}]$

$R_1 \overset{PNO}{\longleftrightarrow}_{\{E\#\}} (R_2 \overset{PNO}{\longleftrightarrow}_{\{E\#\}} R_3)$

$E\#$	$D\#$	*degree*	*salary*
E10	D20	MS	$[120^{\frac{1}{2}}, 100^{\frac{1}{2}}]$

Fig. 12. The results of $(R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2) \xleftrightarrow{PNO}_{\{E\#\}} R_3$ and $R_1 \xleftrightarrow{PNO}_{\{E\#\}} (R_2 \xleftrightarrow{PNO}_{\{E\#\}} R_3)$

- **Case 2:** The Generalized Attribute Derivation (GAD) operation \cite{lim} is used to integrate relations R_1, R_2 and R_3.
  In this case, an attribute derivation function, say F_{salary}, is used in the GAD operation to integrate the values of attribute *salary* in R_1, R_2 and R_3, where $F_{salary}(x,y) = \frac{(x+y)}{2}$. If R_1 and R_2 are first integrated into a temporary relation, then the value of *salary* for "E10" is 110. Then the temporary relation and R_3 are integrated, and the final value of *salary* for integrated tuple "E10" is 105. On the other hand, if R_2 and R_3 are first integrated into a temporary relation, then the value of *salary* for "E10" is 100. Then R_1 and the temporary relation are integrated, and the final value of *salary* for integrated tuple "E10" is 110. These two orders for integrating relations R_1, R_2 and R_3 produce different results since the attribute derivation function F_{salary} is not associative.

From the above discussion, we find that reordering is difficult in approaches that use probabilistic partial values or aggregate functions. However, this problem is both interesting and important, and deserves further research.

Another important issue in future research is semantics analysis of identifying attributes in multiple partial natural outerjoins. For example, consider relations R_1(*E#*, *nickname*, *address*), R_2(*E#*, *Ename*,*age*) and R_3(*Ename*, *nickname*, *phone*) shown in Fig. 13, which record information about employees at a company participating in projects *A*, *B* and *C*, respectively. The key attributes for R_1, R_2, and R_3 are {*E#*}, {*E#*}, and *Ename, nickname*, respectively. We assume that the single attribute *E#* and the attributes {*Ename* and *nickname*} can be used to uniquely identify an employee in the company. Suppose R_1, R_2, and R_3 are integrated as a view *V*(*E#*, *Ename, nickname, address, age,phone*), which contains information about employees participating in project *A*, *B*, or *C*. Intuitively, view *V* can be materialized by means of partial natural outerjoins over R_1, R_2, and R_3. The sets of possible identifying attributes for the pairs <R_1, R_2>, <R_2, R_3>, and <R_1, R_3> are {*E#*}, {*Ename*}, and {*nickname*}, respectively. There are three possible ways to execute the partial natural outerjoins: $(R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2) \xleftrightarrow{PNO}_{\{Ename,nickkname\}} R_3$, $(R_2 \xleftrightarrow{PNO}_{\{Ename\}} R_3) \xleftrightarrow{PNO}_{\{E\#\}} R_1$, and $(R_1 \xleftrightarrow{PNO}_{\{nickname\}} R_3) \xleftrightarrow{PNO}_{\{E\#\}} R_2$. The results are shown in Fig. 14. It can be seen that the result of $(R_1 \xleftrightarrow{PNO}_{\{E\#\}} R_2) \xleftrightarrow{PNO}_{\{Ename,\ nickname\}} R_3$ is desirable since the information for each employee is integrated into a single tuple. The result of $(R_2 \xleftrightarrow{PNO}_{\{Ename\}} R_3) \xleftrightarrow{PNO}_{\{E\#\}} R_1$ is undesirable because employees "E50" and "E60" are both represented by two tuples in the result. We also find that the second and third tuples in the result have incorrect *nickname* and *phone* values for "E50" and "E60," respectively. This is because *Ename* is not a key, which causes the result of $R_2 \xleftrightarrow{PNO}_{\{Ename\}} R_3$ to contain erroneous data. Similarly, the result of $(R_1 \xleftrightarrow{PNO}_{\{nickname\}} R_3) \xleftrightarrow{PNO}_{\{E\#\}} R_2$ is also undesirable. Therefore, the partial natural outerjoin with the key attributes as the identifying attributes should be performed first. The analysis helps the user specify the execution order of partial natural outerjoins. Moreover, the assignment of reliabilities used in the data integration operator deserves further research. Finally, efficient implementations of partial natural outerjoins on a multidatabase system and equivalence transformation rules for a query optimizer are also important issues for future research.

	E#	*nickname*	*address*
R_1	E50	Bob	KF_100
	E60	Cowboy	AM_35

	E#	*Ename*	*age*
R_2	E50	Robert	25
	E60	Robert	26

	Ename	*nickname*	*phone*
R_3	Robert	Bob	3553
	Robert	Cowboy	3906
	Paul	Cowboy	1079

Fig. 13. The relations R_1, R_2, and R_3.

APPENDIX A: BASIC RULES

Let a_i be the associated attribute of selection predicate *p*. Consider the following two cases with respect to the selection.

$(R_1 \overset{PNQ}{\longleftrightarrow}_{\{E\#\}} R_2) \overset{PNQ}{\longleftrightarrow}_{\{Ename,nickname\}} R_3$

E#	*Ename*	*nickname*	*address*	*age*	*phone*
E50	Robert	Bob	KF_100	25	3553
E60	Robert	Cowboy	AM_35	26	3906
~	Paul	Cowboy	~	~	1079

$(R_2 \overset{PNQ}{\longleftrightarrow}_{\{Ename\}} R_3) \overset{PNQ}{\longleftrightarrow}_{\{E\#\}} R_1$

E#	*Ename*	*nickname*	*address*	*age*	*phone*
E50	Robert	Bob	KF_100	25	3553
E50	Robert	[Cowboy$^{\frac{1}{2}}$, Bob$^{\frac{1}{2}}$]	KF_100	25	3906
E60	Robert	[Cowboy$^{\frac{1}{2}}$, Bob$^{\frac{1}{2}}$]	AM_35	26	3553
E60	Robert	Cowboy	AM_35	26	3906
~	Paul	Cowboy	~	~	1079

$(R_1 \overset{PNQ}{\longleftrightarrow}_{\{nickname\}} R_3) \overset{PNQ}{\longleftrightarrow}_{\{E\#\}} R_2$

E#	*Ename*	*nickname*	*address*	*age*	*phone*
E50	Robert	Bob	KF_100	25	3553
E60	Robert	Cowboy	AM_35	26	3906
E60	[Paul$^{\frac{1}{2}}$, Robert$^{\frac{1}{2}}$]	Cowboy	AM_35	26	1079

Fig. 14. The results of different execution orders for performing partial natural outerjoins over R_1, R_2 and R_3.

case 1: $a_i \in P_1$

- **Basic rule (B1):** $\sigma_p(\overline{R_2}) = \phi$
  Proof. Since a_i is a private attribute of R_1, the tuples in $\overline{R_2}$ have null a_i values which cannot satisfy predicate p. □

- **Basic rule (B2):** $\sigma_p(R_1 \overset{PNJ}{\bowtie}_s R_2) = (\sigma_p R_1) \overset{PNJ}{\bowtie}_s R_2$
  Proof. Since a_i is a private attribute of R_1, selection predicate p can be locally performed on R_1 before the partial natural join is executed, which reduces the size of R_1 to be transmitted to the site of R_2. □

- **Basic rule (B3):** $\sigma_p \overline{R_1} = \overline{\sigma_p R_1}$, where $\overline{\sigma_p R_1}$ represents the set of extended unmatched tuples in $\sigma_p R_1$ with respect to $(\sigma_p R_1) \overset{PNJ}{\bowtie}_s R_2$.
  Proof. Let r_1 be the set of tuples which do not satisfy p in the result of $R_1 \overset{PNJ}{\bowtie}_s R_2$ and let r_2 be the set of tuples which do not satisfy p in $\overline{R_1}$. The set of tuples representing the real-world entities which do not appear in the result of $\sigma_p(R_1 \overset{PNJ}{\bowtie}_s R_2)$ in R_1 is

$$U_1 = r_1 \cup r_2 \cup (\sigma_p \overline{R_1}).$$

$r_2 \cup (\sigma_p \overline{R_1})$ is the set of tuples in $\overline{R_1}$. Moreover, the set of tuples representing the real-world entities which do not appear in the result of $(\sigma_p R_1) \overset{PNJ}{\bowtie}_s R_2$, in R_1, is

$$U_2 = \sigma_p \overline{R_1} = \overline{\sigma_p R_1}$$

$r_1 \cup r_2$ is the set of tuples representing the real-world entities which do not satisfy predicate p, in R_1. By rule (**B2**), U_1 must equal U_2. Therefore, $\sigma_p \overline{R_1} = \overline{\sigma_p R_1}$. □

For the case where $a_i \in P_2$, similar rules can be derived.

case 1: $a_i \in S$

- **Basic rule (B4):** $\sigma_p(R_1 \overset{PNJ}{\bowtie}_S R_2) = (\sigma_p R_1) \overset{PNJ}{\bowtie}_S (\sigma_p R_2)$
  Proof. Since a_i is a private attribute, the tuples of R_1 and R_2 which can be joined must have the same a_i values. Therefore, selection predicate p can be locally performed on R_1 and R_2 before the partial natural join is executed, which reduces the sizes of R_1 and R_2. □

  Note that if $a_i \in S_{c'}$, then rule **(B4)** does not apply. For example, consider the query $\sigma_{D\#=D10}(R_1 \overset{PNJ}{\bowtie}_{\{E\#\}} R_2')$, where relations R_1 and R'_2 are shown in Fig. 1 and Fig. 6, respectively. The query result of $\sigma_{D\#=D10}(R_1 \overset{PNJ}{\bowtie}_{\{E\#\}} R_2')$ is (E100,John,D10) while that obtained by executing the modified query is empty. Therefore, the modified query $(\sigma_{D\#=D10} R_1) \overset{PNJ}{\bowtie}_{\{E\#\}} (\sigma_{D\#=D10} R_2')$ is not equivalent to the original one. This is because the tuple (E100,John,~) in R'_2 is eliminated by the subquery $(\sigma_{D\#=D10} R_2')$, which causes the information about employee E100 to be lost.

- **Basic rule (B5):** $\sigma_p \overline{R_1} = \overline{\sigma_p R_1}$
  Proof. The proof is similar to that of rule **(B3).** □

- **Basic rule (B6):** $\sigma_p \overline{R_2} = \overline{\sigma_p R_2}$, where $\overline{\sigma_p R_2}$ represents the set of extended unmatched tuples in $\sigma_p R_2$ with respect to $R_1 \overset{PNJ}{\bowtie}_S (\sigma_p R_2)$.
  Proof. The proof is similar to that of rule **(B3).** □

  Let Q be the set of attributes to be projected. Assume that null tuples are discarded, and that duplicate tuples are not allowed in a relation. Consider the following two cases with respect to the projection.

case 1: $Q \subseteq P_1$

Let $\hat{R}_1$ be the set of tuples in R_1 which match the tuples of R_2 with respect to the operation $R_1 \overset{PNJ}{\bowtie}_S R_2$ and $\tilde{R}_1$ be $R_1 - \hat{R}_1$. $\hat{R}_2$ and $\tilde{R}_2$ are defined similarly. The operation $R_2 \xrightarrow{SJ}_S R_1$ is a *semijoin* from R_2 to R_1 on S, which can be implemented by projecting R_2 on all the attributes in S, shipping the projection to the site where R_1 is located and performing a natural join with R_1.

- **Basic rule (B7):** $\pi_Q(R_1 \overset{PNJ}{\bowtie}_S R_2) = \pi_Q \hat{R}_1 = \pi_Q(R_2 \xrightarrow{SJ}_S R_1)$
  Proof. Since all the attributes to be projected are private attributes of R_1, and since the unmatched tuples of R_1 are not preserved, the operation $\pi_Q(R_1 \overset{PNJ}{\bowtie}_S R_2)$ can be reduced to $\pi_Q \hat{R}_1$. Moreover $\hat{R}_1$, can be obtained by $R_2 \xrightarrow{SJ}_S R_1$. □

$\pi_{E\#,D\#}(R_1 \overset{PNJ}{\bowtie}_{\{E\#\}} R_2')$

$E\#$	$D\#$
E100	D10
E500	[D20$^{\frac{1}{2}}$, D35$^{\frac{1}{2}}$]

$\pi_{E\#,D\#}\ (R_2' \xrightarrow{SJ}_{\{E\#\}} R_1)$

$E\#$	$D\#$
E100	D10
E500	D35

$\pi_{E\#,D\#}\ (R_1 \xrightarrow{SJ}_{\{E\#\}} R_2')$

$E\#$	$D\#$
E100	~
E500	D20

Fig. 15. The results of and $\pi_{E\#,D\#}(R_1 \overset{PNJ}{\bowtie}_{\{E\#\}} R_2'), \pi_{E\#,D\#}(R_2' \xrightarrow{SJ}_{\{E\#\}} R_1)$ and $\pi_{E\#,D\#}(R_1 \xrightarrow{SJ}_{\{E\#\}} R_2')$.

- **Basic rule (B8):** $\pi_Q \overline{R_1} = \pi_Q \tilde{R}_1$
  Proof. Since all the attributes to be projected are private attributes of R_1, the private attributes of R_2 in $\overline{R_1}$ can be ignored.

- **Basic rule (B9):** $\pi_Q \overline{R_2} = \phi$
  Proof. Since the unmatched tuples of R_2 are preserved with null values in the private attributes of R_1, $\pi_Q \overline{R_2}$ is empty. □

For the case where $Q \subseteq P_2$, similar rules can be derived.

case 2: $Q \subseteq S$

- **Basic rule (B10):** $\pi_Q(R_1 \overset{PNJ}{\bowtie}_S R_2) = \pi_Q \hat{R}_1 = \pi_Q(R_2 \xrightarrow{SJ}_S R_1)$
$= \pi_Q \hat{R}_2 = \pi_Q(R_1 \xrightarrow{SJ}_S R_2)$
  Proof. Since all the attributes to be projected are identifying attributes in S, the operation $\pi_Q(R_1 \overset{PNJ}{\bowtie}_S R_2)$ can be reduced to $\pi_Q \hat{R}_1$ or $\pi_Q \hat{R}_2$. Moreover, $\hat{R}_1$ and $\hat{R}_2$ can be obtained by $R_2 \xrightarrow{SJ}_S R_1$ and $R_1 \xrightarrow{SJ}_S R_2$, respectively, according to the definition.
  □

  Note that if $Q \cap S_{c'} \neq \phi$, then rule **(B10)** does not apply. For example, consider the query, $\pi_{E\#,D\#}(R_1 \overset{PNJ}{\bowtie}_{\{E\#\}} R_2')$ where R_1 and R'_2 are shown in Fig. 1 and Fig. 6, respectively. The query result of $\pi_{E\#,D\#}(R_1 \overset{PNJ}{\bowtie}_{\{E\#\}} R_2')$ is not equivalent to that of the modified query $\pi_{E\#,D\#}(R_2' \xrightarrow{SJ}_{\{E\#\}} R_1)$ or $\pi_{E\#,D\#}(R_1 \xrightarrow{SJ}_{\{E\#\}} R_2')$ as shown in Fig. 15. The reason is that the tuples representing the same employee may have different $D\#$ values which cannot be integrated in the processing step for the modified query.
  □

- **Basic rule (B11):** $\pi_Q \overline{R_1} = \pi_Q \tilde{R}_1$
  Proof. Since all the attributes to be projected are identifying attributes in S, the private attributes of R_2 in $\overline{R_1}$ can be ignored. □

- **Basic rule (B12):** $\pi_Q \overline{R_2} = \pi_Q \tilde{R}_2$

Proof. The proof is similar to that of rule **(B11).** □

We consider the case with more than one partial natural outerjoin in the following:

- **Basic rule (B13):** If $B \cap P_2 \neq \phi, \overline{R_1} \overset{PNJ}{\bowtie}_B R_3 = \phi$

  Proof. Since there is at least one private attribute of R_2 in B, and since the unmatched tuples of R_1 are preserved with null values in the private attributes of R_2, the tuples in $\overline{R_1}$ do not match any tuple in R_3. □

  Similarly, if $B \cap P_1 \neq \phi, \overline{R_2} \overset{PNJ}{\bowtie}_B R_3 = \phi$.

REFERENCES

1. ANSI, *Working Draft of SQL2/SQL3*, Technical Report, No. 1, American National Standard Institute, 1992.
2. C. Batini, M. Lenzerini, and S. B. Navathe, “A comparative analysis of methodologies for database schema integration,” *ACM Computing Surveys*, Vol. 18, No. 4, 1986, pp. 323-364.
3. G. Bhargava, P. Goel, and B. Iyer, “Hypergraph based reorderings of outer join queries with complex predicates,” in *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 1995, pp. 304-315.
4. G. Bhargava, P. Goel, and B. Iyer, “Efficient processing of outer joins and aggregate functions,” in *Proceedings of International Conference on Data Engineering*, 1996, pp. 441-449.
5. Y. Breitbart, P. L. Olson, and G. R. Thompson, “Database integration in a distributed heterogeneous database system,” in *Proceedings of International Conference on Data Engineering*, 1986, pp. 301-310.
6. Y. Breitbart, “Multidatabase interoperability,” *ACM SIGMOD Record*, Vol. 19, No. 3, 1990, pp. 53-60.
7. A. L. P. Chen, “Outerjoin optimization in multidatabase systems,” in *Proceedings of International Symposium on Databases in Parallel and Distributed Systems*, 1990, pp. 211-218.
8. E. F. Codd, “Missing information (applicable and inapplicable) in relational databases,” *ACM SIGMOD Record*, Vol. 15, No. 4, 1986, pp. 53-78.
9. C. J. Date, “The outer join, chapter 12” *Relational Database Selected Writings*, Addison Wesley, 1986.
10. U. Dayal, “Processing queries over generalized hierarchies in a multidatabase systems,” in *Proceedings of the International Conference on Very Large Data Bases*, 1983, pp. 342-353.
11. U. Dayal and H. Y. Hwang, “View definition and generalization for database integration in a multidatabase system,” *IEEE Transactions on Software Engineering*, Vol. 10, No. 6, 1984, pp. 628-644.
12. M. M. David, “Advanced capabilities of the outer join,” *ACM SIGMOD Record*, Vol. 21, No. 1, 1992, pp. 65-70.
13. S. M. Deen, R. R. Amin, and M. C. Taylor, “Data integration in distributed databases,”

IEEE Transactions on Software Engineering, Vol. 13, No. 7, 1987, pp.860-864.
14. L. G. DeMichiel, "Resolving database incompatibility: an approach to performing relational operations over mismatched domains," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 1, No. 4, 1989, pp. 485-493.
15. C. A. Galindo-Legaria and A. Rosenthal, "How to extend a conventional optimizer to handle one- and two-sideed outerjoin," in *Proceedings of the International Conference on Data Engineering*, 1992, pp. 402-409.
16. C. A. Galindo-Legaria, "Outerjoins as disjunctions," in *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 1994, pp. 348-358.
17. C. A. Galindo-Legaria and A. Rosenthal, "Outerjoin simplification and reordering for query optimization," *ACM Transactions on Database Systems*, Vol. 22, No. 1, 1997, pp. 43-74.
18. J. L. Koh and A. L. P. Chen, "Integration of heterogeneous object schemas," in *Proceedings of the International Conference on Entity-Relationship Approach*, 1993, pp. 289-300.
19. B. S. Lee and G. Wiederhold, "Outer joins and filters for instantiating objects from relational databases through views," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 6, No. 1, 1994, pp. 108-119.
20. E. P. Lim, J. Srivastava, and S. Y. Hwang, "An algebraic transformation framework for multidatabase queries," *Distributed and Parallel Databases*, Vol. 3, No. 1, 1995, pp. 273-307.
21. W. Litwin, A. Abdellatif, A. Zeroual, and B. Nicolas, "MSQL: a multidatabase language," *Information Science*, Vol. 49, No. 1, 1989, pp. 59-101.
22. A. Motro, "Superviews: virtual integration of multiple databases," *IEEE Transactions on Software Engineering*, Vol. 13, No. 7, 1987, pp. 785-798.
23. H. Pirahesh, C. Mohan, and J. Cheng, "Sequential and parallel algorithms for unified execution of outerjoin and subqueries," IBM Almaden Research Center, 1993.
24. A. Rosenthal and C. A. Galindo-Legaria, "Query graphs, implementing trees, and freely-reorderable outerjoins," in *Proceedings of ACM-SIGMOD International Conference on Management of Data*, 1990, pp. 291-299.
25. P. S. M. Tsai and A. L. P. Chen, "Querying uncertain data in heterogeneous databases," in *Proceedings of IEEE International Workshop on Research Issues on Data Engiennring: Interoperability in Multidatabase Systems*, 1993, pp. 161-168.
26. 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.

Pauray S. M. Tsai (½²¨ q° ¡) received the BS and MS degrees in computer science information engineering from National Chiao Tung University, Taiwan, in 1990 and 1992, respectively, and the Ph.D. degree in computer science from National Tsing Hua University, Taiwan, in 1996. Currently she is an associate professor in the Department of Information Management, Ming Hsin Institute of Technology, Taiwan. Her research interests include multidatabase systems, object-oriented database systems, and data mining.

Arbee L. P. Chen ([illegible]) received a Ph.D. degree in computer engineering from the University of Southern California in 1984. He is currently a professor at National Tsing Hua University after being affiliated with Unisys and Bellcore. He organized (and served as a program co-chair of) 1995 IEEE Data Engineering Conference, COOPIS '97, and DASFAA'99. His research interests include multimedia databases, data mining, and mobile computing. He is a recipient of the NSC Distinguished Research Award.