OUTERJOIN OPTIMIZATION IN MULTIDATABASE SYSTEMS

Arbee L.P. Chen

Bell Communications Research
Piscataway, NJ 08854, U.S.A.
rb@ctt.bellcore.com

ABSTRACT

Outerjoin is used in distributed relational multidatabase systems for integrating local schemas to a global schema. Queries against the global schema need to be modified, optimized, and decomposed into subqueries at local sites for processing. Since outerjoin combines local relations in different databases to form a global relation, it is expensive to process. In this paper, based on the structure of the query and the definition of the schemas, queries with outerjoin, join, select and project operations are optimized. Conditions where outerjoin can be avoided or be transformed into a one-side outerjoin are identified. By considering these conditions the response time for query processing can be reduced.

1. Introduction

It is becoming increasingly important to manage databases as a repository resource and allow application programs to access this resource in a heterogeneous distributed environment. A *multidatabase* system [25], [28] is a system to meet this goal by providing uniform and integrated access to a distributed collection of existing databases.

Views are defined in multidatabase systems to provide data distribution and schema difference transparencies [5], [18], [19], [20]. To process queries against these views, modification of queries [27] from against the view to against the underlying local schemas is needed.

Many distributed query processing algorithms have been proposed. Some make use of *semijoins* to reduce the amount of data transfer [1], [3], [4], [9] - [12], [29]; some make use of the *fragment and replicate* strategy to allow parallel processing [30], [31]; some integrate these two strategies for adapting to various network environments [8], [32]; some use *operation grouping* as the heuristic [14]; some consider query transformation for multidatabases [17], [22]; moreover, some apply semantic information for fast query processing [21], [23], [24], [33].

In this paper, query optimization in a relational multidatabase system is considered. Outerjoin is used for integrating local schemas to provide views of the multidatabase. After the query modification process, a query could contain outerjoin, join, select, and project operations. Based on the structure of the query and the definition of the schemas, the modified queries are optimized. None of the above query optimization algorithms consider the optimization of outerjoin processing.

The paper is organized as follows. Section 2 discusses schema integration by outerjoin. The definition of outerjoin and its use for schema integration in relational multidatabases are detailed. Query modification and the semantics of the modified queries are discussed in Section 3. Section 4 provides the query processing strategy for multidatabase queries. Section 5 concludes this work.

2. Schema Integration by Outerjoin

2.1 Outerjoin

The join (or innerjoin) over two relations results in a new relation in which each tuple is formed by joining two tuples, one from each of the original relation, such that the joining attribute values of the two tuples satisfy the join predicate. Therefore, it can lose information in the sense that unmatched tuples in the joining relations do not participate in the result of the join. In contrast, the outerjoin does not lose such information. An outerjoin is obtained by appending additional tuples to the result of the corresponding join. The additional tuples are those unmatched tuples, extended with null values for the other attributes.

The formal definition of outerjoin as given in Date [16] is as follows. Let $R(A,B_1)$ and $S(B_2,C)$ be two relations with attributes $R.A$, $R.B_1$, $S.B_2$, and $S.C$. Define X to be the θ-join of R on B_1 with S on B_2, where θ represents any valid scalar comparison operator;

$$X = J_{R.B_1 \theta S.B_2}(R, S)$$

where $J_{R.B_1 \theta S.B_2}$ denotes the θ-join. Define R' and S' as follows:

$$R' = R - X[A,B_1]$$
$$S' = S - X[B_2,C]$$

Here $X[A,B_1]$ and $X[B_2,C]$ are the projection of X on A and B_1 and the projection of X on B_2 and C, respectively. R' and S' are therefore the unmatched tuples of R and S, respectively, with respect to the θ-join. The outerjoin of R on B_1 with S on B_2, denoted $OJ_{R.B_1 \theta S.B_2}$, is defined as:

$$OJ_{R.B_1 \theta S.B_2} = X \cup (R' \times (-,-)) \cup ((-,-) \times S')$$

CH2895-1/90/0000/0211$01.00 © 1990 IEEE

where "-" denotes the null value, and "×" denotes the extended Cartesian product. The left and right outerjoins (or *one-side outerjoins*) of R on B_1 with S on B_2, denoted $\mathbf{LOJ}_{R.B_1 \theta S.B_2}$ and $\mathbf{ROJ}_{R.B_1 \theta S.B_2}$, are defined as:

$$\mathbf{LOJ}_{R.B_1 \theta S.B_2} = X \cup (R' \times (-,-))$$
$$\mathbf{ROJ}_{R.B_1 \theta S.B_2} = X \cup ((-,-) \times S')$$

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

If θ is equality, the θ-join is referred to as an equi-join (which applies to both innerjoins and outerjoins).

2.2 Use of Outerjoin for Schema Integration

Since an outerjoin preserves information for the two outerjoining relations, it can therefore be used to "union" two "semantically related" relations in a multidatabase system [5], [14], [19]. The two outerjoining relations are *local relations* in different databases while the result of the outerjoin is a *global relation* in the multidatabase.

In the relational data model, a key is an identifier of a relation and a non-key attribute represents a property of the relation, functionally determined by the key. For two local relations in different databases to be qualified as "semantically related" relations, we require that their keys be semantically the same. That is, the keys may have different names or be represented in different formats, conceptually they represent the same thing. In order to integrate these two semantically related local relations, *natural outerjoin* over these two relations is performed. A natural outerjoin is an equi-outerjoin on the common attributes with one set of the common attributes projected out of the resultant relation. (Notice that there are two sets of the joining attribute in the resultant relation of a join, one from each joining relation.) The common attributes are then designated as the *global relation identifier*.

The data inconsistency problem in the schema integration will not be discussed in this paper. That is, if the key values are the same in two tuples each from one local relation, the values of the other common attributes in the two tuples are assumed the same.

Natural outerjoin (we use outerjoin and its notation hereafter for natural outerjoin) as a schema integration operation generates two groups of tuples in the global relation when it is materialized. One group consists of the tuples whose identifier values can be found in both local relations. The values of the other attributes in the tuples come from the values of the non-outerjoining attributes in both local relations. This group consists of "objects" which exist in both local relations, and whose properties are stored in both local relations.

The other group consists of the tuples whose identifier values can only be found in one relation, say R_1, but not in the other, say R_2. The values of the other attributes in the tuples are from the values of the non-outerjoining attributes in R_1 and null for the others. This group consists of objects which exist in one local relation only. The properties of these objects are therefore partly stored in the local relation that they belong to and partly unknown.

Therefore, outerjoin represents the generalization concept [26] in semantic data modeling, which has been used in the extended relational model RM/T [13]. Notice that when both local relations contain the same set of attributes, the global relation is actually the union of the tuples in the local relations with the redundant tuples eliminated. Thus, the local relations can be conceptually considered as fragments of a horizontally partitioned global relation. When there exist duplicate key values in the local relations, they can be considered as fragments of a vertically partitioned global relation [7].

Example 1: Two databases DB_1 and DB_2, each contains two relations: IEEE(SS#,Name) and Subs(SS#,Journal) for DB_1 and ACM(SS#,Age) and Subscribe(SS#,Journal) for DB_2, are to be integrated as a multidatabase. The global relations MEMBER and SUBS are defined as follows:

MEMBER(SS#,Name,Age) ≡ $\mathbf{OJ}_{SS\#}$ (IEEE, ACM)
SUBS(SS#,Journal) ≡ $\mathbf{OJ}_{SS\#,Journal}$ (Subs, Subscribe)

where the subscript of **OJ** represents the common attributes.

The contents of the local relations IEEE, Subs, ACM, Subscribe, and the global relations MEMBER, SUBS (when materialized) are shown in Figure 1.

3. Query Modification

3.1 Modification process

An expression of relational algebra can be interpreted not only as the specification of the semantics of a query, but also as the specification of a sequence of operations [6]. For example, a query to retrieve the names of the computer society members who subscribe to "TODS" in the example multidatabase can be expressed in relational algebra as follows.

$$\pi_{Name}\ \sigma_{Journal=TODS}\ \mathbf{J}_{SS\#}\ (\text{MEMBER, SUBS})$$

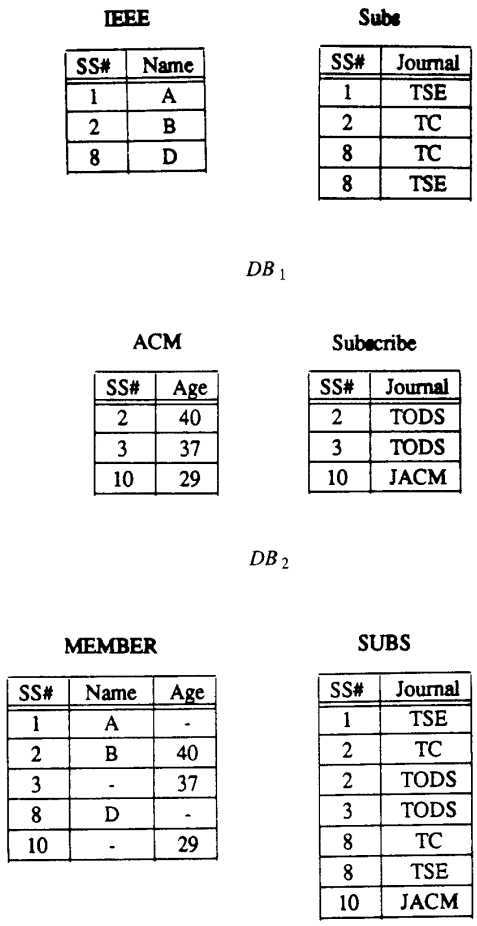
where π denotes project, σ denotes select and $\mathbf{J}_{SS\#}$ denotes equi-join on the attribute SS#. MEMBER and SUBS are global relations and thus need to be replaced by their schema definitions. After the replacement, the modified query is as follows.

$$\pi_{Name}\ \sigma_{Journal=TODS}\ \mathbf{J}_{SS\#}\ (\mathbf{OJ}_{SS\#}\ (\text{IEEE, ACM}),\ \mathbf{OJ}_{SS\#,Journal}\ (\text{Subs, Subscribe}))$$

Attribute names in the user query may also need to be mapped to names in the local relations.

3.2 Semantics of the modified query

Since null values may exist in the global relations, operations on the null values as well as null values in the answer are possible. Two algebraic operators MAYBE_SELECT and MAYBE_JOIN were defined in Date [15] and Codd [13]. MAYBE_SELECT selects tuples for which the value of a specified attribute is null, and MAYBE_JOIN joins tuples for which the value of either of the joining attributes is null.



IEEE

SS#	Name
1	A
2	B
8	D

Subs

SS#	Journal
1	TSE
2	TC
8	TC
8	TSE

DB_1

ACM

SS#	Age
2	40
3	37
10	29

Subscribe

SS#	Journal
2	TODS
3	TODS
10	JACM

DB_2

MEMBER

SS#	Name	Age
1	A	-
2	B	40
3	-	37
8	D	-
10	-	29

SUBS

SS#	Journal
1	TSE
2	TC
2	TODS
3	TODS
8	TC
8	TSE
10	JACM

The Multidatabase

Figure 1: local relations and global relations.

However, these operators will not be considered in the discussion of the query processing in the next section. That is, null values are not involved in either select or join operation in the discussion.

However, the null values in the answer have their meanings, and have to be considered. For example, if there exist null values in the answer of the above query, it means that there are members in the multidatabase, who subscribe to "TODS" but their names are not stored. The number of the null values may also be important because it represents the number of such members in the multidatabase.

As the modified algebraic expression of the query shows, outerjoin needs to be processed before join and select.

However, outerjoin combines all data in local relations, which is expensive and needs to be optimized. Outerjoin is a system-specified operation, which "prepares" the global relations for processing the user query. In certain situations it need not be processed for answering the user query. In other situations, a one-side outerjoin is enough. The query optimization techniques to be discussed in the following identify these situations based on the structure of a query and the locality of data in the multidatabase.

4. Distributed Query Processing

After query modification, four operators may exist in a query. They are select, project, join and outerjoin. We will emphasize the optimization of the outerjoin processing to reduce query processing and data transmission costs.

4.1 Assumptions and definitions

To simplify the discussion, a multidatabase system consisting of two databases is used to illustrate the concepts. Each database contains a set of relations, and is assumed to reside at a site. There exists at least one relation in each database which is involved in the integration of the two databases. The integration of two relations is by defining a global relation scheme as an outerjoin over the common attributes of these two relations. Based on the schemas, some definitions are as follows:

R_{ij}: local relation i at site j.
A_{ij}: attributes of R_{ij}.
C_i: the common attributes of R_{i1} and R_{i2}, where R_{i1} and R_{i2} are semantically related.
C: the union of all C_i's in the multidatabase.
R_i: the global relation produced by outerjoining R_{i1} and R_{i2}.
Q_{ij}: attributes in A_{ij} - C_i, which are common to some attributes in relation R_{st}, where $s \neq i$ and $t \neq j$ (i.e., $Q_{ij} = (A_{ij} - C_i) \cap A_{st}$).
P_{ij}: *local attributes* of R_{ij}, which are not common to any attributes in other databases; $P_{ij} = A_{ij} - C_i - Q_{ij}$.
P^j: all local attributes at site j; $P^j = \bigcup_i P_{ij}$.

Notice that we use single subscript for information regarding a global relation, and single superscript for a site. Also, A_{ij} is divided into three *segments*. One contains the outerjoining attribute which is used for constructing global relations; another contains other common attributes with attributes in other databases for possible join operation; and the other contains local attributes which are not common to attributes in other databases. Based on the user query, some more definitions follow.

The select clauses considered are of the form: "Y θ constant," where Y is called a *select attribute* and θ represents any valid scalar comparison operator.

S: the set of select attributes in a query.
T: the set of target attributes in a query.

Example 2: Assume database 1 at site 1 contains two relations:

R_{11}(I, J, K), where I is the key
R_{21}(L, I, M), where L is the key

and database 2 at site 2 contains:

R_{12}(I, S), where I is the key
R_{22}(L, S, K), where L is the key

R_{11} and R_{12} can be integrated to form a global relation R_1(I, J, K, S). Similarly, R_{21} and R_{22} can be integrated to form R_2(L, I, M, S, K). Then $C_1 = \{I\}$, $C_2 = \{L\}$, $Q_{11} = Q_{22} = \{K\}$, $Q_{21} = \{I\}$, $P^1 = \{J, M\}$ and $P^2 = \{S\}$.

Notice that I and S in database 1 and database 2, respectively, can be used to join the two relations in the database. These attributes also exist in both global relations R_1 and R_2, and thus can be used for join operations. Moreover, by integrating the two databases, attribute K is identified as a common attribute between R_{11} and R_{22}, which can therefore also be used to join the two global relations.

4.2 Query processing strategy

Define *answer tuple identifiers* as the values of the global relation identifier in the tuples which contain values for the answer. The query processing strategy for queries with qualification clauses consists of two steps (queries without qualification will be discussed in Section 4.4):

Multidatabase Query Processing Algorithm:

Step 1: Compute the answer tuple identifiers.
Step 2: Based on the answer tuple identifiers, compute the answer.

The answer tuple identifiers can be computed by processing the qualification clauses (i.e., selects and joins) in the query. Based on the distribution of the target attributes in the local relations, outerjoins and data concatenation may be required to get the answer after the answer tuple identifiers are computed. ORION's query processing strategy [2] also follows this two-step strategy, i.e., it retrieves object UID's (unique identifiers) first then based on the UID, it retrieves the values of interest from the object.

In the following, the query processing is discussed based on the distribution of the target attributes in the segments of local relations. In Section 4.3, all the target attributes are assumed local attributes at a site. In Section 4.4, we assume that all the target attributes are contained in the outerjoining attributes. For these two cases, we assume there exist qualification clauses in the query. We discuss other situations in Section 4.5, which include queries with other target attribute distributions and queries which do not have any qualification clause. A property which is useful for the discussion follows:

Property 1: For a query with qualification clauses, if a local relation R_{ij} does not contain select or join attributes, and it either contains target attributes in C_i or does not contain any target attribute, then it need not be involved in the query processing for producing the answer.

Proof: Since R_{ij} does not contain select or join attributes, none of the answer tuple identifiers could be obtained from R_{ij}. That is, step 1 of the multidatabase query processing algorithm does not apply on R_{ij}. If R_{ij} does not contain target attributes, then step 2 does not apply either. Now, assume that C_i contains target attributes. In the global relation R_i, the tuples which have the values of C_i in $R_{ij}.C_i$ - $R_{ik}.C_i$, where $k \neq j$, have null values for the select or join attributes (a global relation must contain either select or join attributes). This is because R_{ij} does not contain select or join attributes. Since the MAYBE_SELECT and MAYBE_JOIN are not supported, these tuples have no effects on the answer. That is, R_{ij} need not be involved in the query processing. □

4.3 Target attributes are local attributes

Assume all the target attributes are local attributes at site 1, that is, $T \subseteq P^1$. The processing of select and join clauses will be discussed separately. When there exist select and join clauses, select processing can be considered first then join processing.

4.3.1 select processing

Select is a unary operation. That is, only one relation is involved in a select operation. $T \subseteq P^1$ implies $T \subseteq P_{i1}$ when R_i is the relation which contains the select attributes. We discuss the select processing for the following cases.

- $S \subseteq P_{i1} \cup Q_{i1}$.

  By Property 1, R_{i2} is not involved in the processing of this query. Therefore, the query can be processed at site 1, and no outerjoin processing is needed. The query transformation for queries with a single select clause is as follows. For queries with more than one select clauses, it is easy to extend the transformation process.

  $\pi_T\ \sigma_{P_{i1} \cup Q_{i1}}\ \mathbf{OJ}_{C_i}\ (R_{i1}, R_{i2})$
  $= \pi_T\ \sigma_{P_{i1} \cup Q_{i1}}\ R_{i1}$

  where $\sigma_{P_{i1} \cup Q_{i1}}$ denotes the select clause with the select attribute in $P_{i1} \cup Q_{i1}$.

- $S \subseteq P_{i2} \cup Q_{i2}$

  The select attribute is in $P_{i2} \cup Q_{i2}$ while the target attributes are in P_{i1}. The global relation identifier C_i relates them in the local relations R_{i1} and R_{i2}. (Actually, the common key, which is contained in C_i, is enough for relating the select attribute and the target attributes; however, we use C_i for the consistency of the discussion for other cases.) To process this query, in step 1 we perform select at site 2 over R_{i2} to obtain the answer tuple identifiers. These answer tuple identifiers are then used in step 2 for a one-side outerjoin with R_{i1} to get the answer. This outerjoin generates null target attribute values for those answer tuple identifiers which do not match with the values in $R_{i1}.C_i$. These null values represent the existence of tuples in R_{i2}, which satisfy the select qualification but do not have the target attribute values (since $T \subseteq P_{i1}$).

  The query transformation is shown as follows. This transformation saves both transmission cost and processing cost. Instead of transmitting a relation, only the answer

tuple identifiers need to be transmitted. Also, instead of processing an outerjoin of two relations, only one-side outerjoin of one relation and the outerjoining attribute in the other relation is needed.

$\pi_T\ \sigma_{P_{i2} \cup Q_{i2}}\ \mathbf{OJ}_{C_i}\ (R_{i1}, R_{i2})$
$= \pi_T\ \mathbf{LOJ}_{C_i}\ (\pi_{C_i}\ \sigma_{P_{i2} \cup Q_{i2}}\ R_{i2}, R_{i1})$

- $S \subseteq C_i$

  In this case, the select is performed in parallel at both sites. $R_{i2}.C_i$ is then projected and sent to site 1 for outerjoin processing. Notice that this outerjoin is not a one-side outerjoin as in the above case since the select is on C_i and thus the answer tuple identifiers could exist at both sites. The query transformation is shown in the following.

$\pi_T\ \sigma_{C_i}\ \mathbf{OJ}_{C_i}\ (R_{i1}, R_{i2})$
$= \pi_T\ \mathbf{OJ}_{C_i}\ (\sigma_{C_i}\ R_{i1}, \pi_{C_i}\ \sigma_{C_i}\ R_{i2})$

4.3.2 join processing

A join clause joins two relations on the joining attributes. Based on where the joining attributes reside, the join processing is discussed. Denote J_{ij} as the joining attribute in R_{ij}. When the joining attribute is in C_i, it is denoted J_i. Also, assume that R_i and R_j are both global relations formed by integrating local relations in the two databases.

- $J_{i1}, J_{j1} \in P^1$

  By Property 1, only R_{i1} and R_{j1} are involved in the query processing. That is, the query can be processed at site 1, and no outerjoin processing is needed. The query transformation is as follows.

$\pi_T\ \mathbf{J}_{P^1}\ (\mathbf{OJ}_{C_i}\ (R_{i1}, R_{i2}), \mathbf{OJ}_{C_j}\ (R_{j1}, R_{j2}))$
$= \pi_T\ \mathbf{J}_{P^1}\ (R_{i1}, R_{j1})$

  where $\mathbf{J}_{P^1}$ denotes the join clause with the joining attributes in P^1.

- $J_{i2}, J_{j2} \in P^2$

  The joining attributes are in P^2, but the target attributes are in P^1. Similar to the select processing, in step 1 the join is processed at site 2 to get the answer tuple identifiers. The answer tuple identifiers may come from C_i if the target attributes are in P_{i1} or from C_j if the target attributes are in P_{j1} or from C_i and C_j if the target attributes are in both P_{i1} and P_{j1}. In step 2, one-side outerjoin is processed at site 1 to compute the answer based on these answer tuple identifiers. If the target attributes are in both P_{i1} and P_{j1}, two one-side outerjoins are needed, one for each relation. Concatenation of the target attributes from the results will produce the answer. The concatenation of two attributes with the same cardinality is done by concatenating the values in the two attributes tuple by tuple. The query transformation is shown as follows.

$\pi_T\ \mathbf{J}_{P^2}\ (\mathbf{OJ}_{C_i}\ (R_{i1}, R_{i2}), \mathbf{OJ}_{C_j}\ (R_{j1}, R_{j2}))$
$= \pi_{T \cap P_{i1}}\ \mathbf{LOJ}_{C_i}\ (\pi_{C_i}\ \mathbf{J}_{P^2}\ (R_{i2}, R_{j2}), R_{i1})$
$\| \pi_{T \cap P_{j1}}\ \mathbf{LOJ}_{C_j}\ (\pi_{C_j}\ \mathbf{J}_{P^2}\ (R_{i2}, R_{j2}), R_{j1})$

  where $\|$ denotes concatenation. Notice that when $T \cap P_{k1} = \emptyset$, k = i or j, the evaluation of the associated outerjoin is not needed.

- $J_{i1} \in Q_{i1}, J_j \in C_j$

  By Property 1, R_{i2} is not involved in the query processing. $R_{j2}.C_j$ which contains the joining attribute $R_{j2}.J_j$ is sent to site 1 and unioned with $R_{j1}.C_j$ for the join processing. The resultant C_j is then outerjoined with R_{j1} when the target attributes are in P_{j1}. If the target attributes are in both P_{i1} and P_{j1} then the concatenation of the target attributes is needed. The transformation is shown in the following.

$\pi_T\ \mathbf{J}_{C_j}\ (\mathbf{OJ}_{C_i}\ (R_{i1}, R_{i2}), \mathbf{OJ}_{C_j}\ (R_{j1}, R_{j2}))$
$= \pi_{T \cap P_{i1}}\ \mathbf{J}_{C_j}\ (R_{i1}, \cup\ (\pi_{C_j}\ R_{j1}, \pi_{C_j}\ R_{j2}))$
$\| \pi_{T \cap P_{j1}}\ \mathbf{LOJ}_{C_j}\ (\pi_{C_j}\ \mathbf{J}_{C_j}\ (R_{i1}, \cup\ (\pi_{C_j}\ R_{j1}, \pi_{C_j}\ R_{j2})), R_{j1})$

  Notice that when all the target attributes are in R_{i1}, no outerjoin is needed. However, when the target attributes are in both relations the query processing strategy may generate execution plans which need more processing time. Similar situations can be seen in other cases. The goal of this approach is to identify conditions where outerjoin can be optimized.

- $J_{i2} \in Q_{i2}, J_j \in C_j$

  Similar to the above case, $R_{j1}.C_j$ can be sent to site 2 for the join processing. The resultant answer tuple identifiers C_i and/or C_j are then sent back to site 1 for one-side outerjoin processing. Whether to send C_k, k = i or j, depends on the existence of the target attributes in R_{k1}. Concatenation of the outerjoin results is needed when the target attributes are in both P_{i1} and P_{j1}. The transformation is shown in the following.

$\pi_T\ \mathbf{J}_{C_j}\ (\mathbf{OJ}_{C_i}\ (R_{i1}, R_{i2}), \mathbf{OJ}_{C_j}\ (R_{j1}, R_{j2}))$

$= \pi_{T \cap P_{i1}}\ \mathbf{LOJ}_{C_i}\ (\pi_{C_i}\ \mathbf{J}_{C_j}\ (R_{i2}, \cup\ (\pi_{C_j}\ R_{j1}, \pi_{C_j}\ R_{j2})), R_{i1})$
$\| \pi_{T \cap P_{j1}}\ \mathbf{LOJ}_{C_j}\ (\pi_{C_j}\ \mathbf{J}_{C_j}\ (R_{i2}, \cup\ (\pi_{C_j}\ R_{j1}, \pi_{C_j}\ R_{j2})), R_{j1})$

- $J_i \in C_i, J_j \in C_j$

  This query can be processed by sending $R_{i2}.C_i$ and $R_{j2}.C_j$ to site 1 for the join processing. The resultant answer tuple identifiers C_i and/or C_j are then used for one-side outerjoin processing. Concatenation of the outerjoin results is needed when the target attributes are in both P_{i1} and P_{j1}. The transformation is shown in the following.

$\pi_T\ \mathbf{J}_{C_i}\ (\mathbf{OJ}_{C_i}\ (R_{i1}, R_{i2}), \mathbf{OJ}_{C_j}\ (R_{j1}, R_{j2}))$
$= \pi_{T \cap P_{i1}}\ \mathbf{LOJ}_{C_i}\ (\pi_{C_i}\ \mathbf{J}_{C_i}\ (\cup\ (\pi_{C_i}\ R_{i1}, \pi_{C_i}\ R_{i2}),$
$\quad \cup\ (\pi_{C_j}\ R_{j1}, \pi_{C_j}\ R_{j2})), R_{i1})$
$\| \pi_{T \cap P_{j1}}\ \mathbf{LOJ}_{C_j}\ (\pi_{C_j}\ \mathbf{J}_{C_i}\ (\cup\ (\pi_{C_i}\ R_{i1}, \pi_{C_i}\ R_{i2}),$
$\quad \cup\ (\pi_{C_j}\ R_{j1}, \pi_{C_j}\ R_{j2})), R_{j1})$

- $J_{i1} \in Q_{i1}, J_{j2} \in Q_{j2}$

  This query can be processed by sending $R_{j2}.J_{j2}$ to site 1 for the join processing. If P_{j1} contains target attributes then $R_{j2}.C_j$ has to be sent with $R_{j2}.J_{j2}$ for computing the answer tuple identifiers to perform the one-side outerjoin with R_{j1}. Concatenation of the results is needed when the target attributes are in both P_{i1} and P_{j1}. The transformation is shown in the following.

  $\pi_T \mathbf{J}_{Q_{i1}} (\mathbf{OJ}_{C_i} (R_{i1}, R_{i2}), \mathbf{OJ}_{C_j} (R_{j1}, R_{j2}))$
  $= \pi_T \cap_{P_{i1}} \mathbf{J}_{Q_{i1}} (R_{i1}, \pi_{C_j, Q_{j2}} R_{j2})$
  $\| \pi_T \cap_{P_{j1}} \mathbf{LOJ}_{C_j} (\pi_{C_j} \mathbf{J}_{Q_{i1}} (R_{i1}, \pi_{C_j, Q_{j2}} R_{j2}), R_{j1})$

4.4 Target attributes are contained in outerjoining attributes

We consider T $\subseteq$ C in this subsection. As in Section 4.3, the select processing and join processing are discussed in cases.

4.4.1 select processing

- $S \subseteq P_{i1} \cup Q_{i1}$.

  By Property 1, R_{i2} is not involved in the processing of this query. Therefore, the query can be processed at site 1, and no outerjoin processing is needed. The query transformation is as follows.

  $\pi_T \sigma_{P_{i1} \cup Q_{i1}} \mathbf{OJ}_{C_i} (R_{i1}, R_{i2})$
  $= \pi_T \sigma_{P_{i1} \cup Q_{i1}} R_{i1}$

- $S \subseteq P_{i2} \cup Q_{i2}$

  By Property 1, R_{i1} is not involved in the query processing. The query can be processed at site 2. No outerjoin is needed.

- $S \subseteq C_i$

  In this case, only C_i is involved in the query processing. Outerjoin can be replaced by a union. The query can be processed by performing select and project in parallel at both sites, then unioning the results. The query transformation is as follows.

  $\pi_T \sigma_{C_i} \mathbf{OJ}_{C_i} (R_{i1}, R_{i2})$
  $= \cup (\pi_T \sigma_{C_i} R_{i1}, \pi_T \sigma_{C_i} R_{i2})$

4.4.2 join processing

- $J_{i1}, J_{j1} \in P^1$

  The query can be processed at site 1, by Property 1. No outerjoin is needed.

- $J_{i2}, J_{j2} \in P^2$

  This case is similar to the above one. The query can be processed at site 2. No outerjoin is needed.

- $J_{i1} \in Q_{i1}, J_j \in C_j$

  By Property 1, R_{i2} is not involved in the query processing. To process this query, we send $R_{j2}.C_j$ to site 1, perform a union with $R_{j1}.C_j$, join the result of the union with R_{i1}, then project on the target attributes to get the answer. No outerjoin is needed. The query transformation is as follows.

  $\pi_T \mathbf{J}_{C_j} (\mathbf{OJ}_{C_i} (R_{i1}, R_{i2}), \mathbf{OJ}_{C_j} (R_{j1}, R_{j2}))$
  $= \pi_T J_{C_j} (R_{i1}, \cup (\pi_{C_j} R_{j1}, \pi_{C_j} R_{j2})$

- $J_{i2} \in Q_{i2}, J_j \in C_j$

  The query processing is similar to the above case. No outerjoin is needed.

- $J_i \in C_i, J_j \in C_j$

  The query processing is similar to the case with the same condition on the joining attributes in Section 4.3.2. However, no outerjoin is needed here since the target attributes are in C.

- $J_{i1} \in Q_{i1}, J_{j2} \in Q_{j2}$

  Again, the query processing is similar to the case with the same condition on the joining attributes in Section 4.3.2. Since the target attributes are in C, no outerjoin is needed.

4.5 Other queries

4.5.1 queries with qualification clauses

The same query processing algorithm can be applied for queries with other target attribute distributions. The answer tuple identifiers are computed in step 1 by processing the qualification clauses. Then, in step 2, these identifiers are used for outerjoin operations. Concatenation of the results is needed when the target attributes span more than one local relations in either the same database or different databases.

4.5.2 queries without qualification

If the target attributes are in $P_{ik} \cup Q_{ik}$, k = 1 or 2, then only one site is involved in the query processing. If the target attributes are in C_i then the target attribute values from both C_{i1} and C_{i2} need to be retrieved and unioned. In other cases, outerjoin is needed.

5. Conclusions

We have described query processing strategies in a relational multidatabase system. Outerjoin is used to integrate local relations in different databases to form a global relation. A query has to be modified from against the global schema to the local schemas for processing. Outerjoin thus exists in the modified query. The processing of outerjoin is expensive since it combines two relations. However, in some cases, it need not be processed for producing the answer for the query.

An approach to optimize outerjoin processing was proposed. Attributes in a local relation are divided into three segments. One contains the outerjoining attribute which is used for constructing global relations; another contains other common attributes with attributes in other databases for possible join operation; and the other contains local attributes which are not common to attributes in other databases. Based on the allocation of the target attributes, select attributes and joining attributes in these segments, the multidatabase query is processed. The query processing strategy computes the answer

tuple identifiers first by processing the qualification clauses in the query. Outerjoin is then performed when needed to compute the answer based on these answer tuple identifiers.

Concatenation of the results from outerjoins is needed when the target attributes span more than one local relations in either the same database or different databases.

The situations in which local query processing cost and data transmission cost can be reduced were identified by this approach. When all the target attributes are contained in the outerjoining attributes, no outerjoin processing is needed. When all the target attributes are local attributes at a site then in some cases outerjoin processing can also be avoided. When it is needed, it may be changed to a one-side outerjoin of a relation and the outerjoining attribute of the other relation, which requires less processing costs.

Some research issues are currently under investigation, including consideration of data inconsistency, data fragmentation, and MAYBE_SELECT and MAYBE_JOIN operations, incorporation of semijoin strategy with the two-step strategy, and derivation of a general algebraic framework for outerjoin optimization.

Acknowledgement: I am grateful to the reviewers, Arnie Rosenthal and Gomer Thomas for many useful suggestions.

REFERENCES

[1] Apers, P., A. Hevner, S.B. Yao, "Optimization algorithm for distributed queries," *IEEE Transactions on Software Engineering,* January 1983.

[2] Banerjee, J., W. Kim, K.C. Kim, "Queries in object-oriented databases," Proc. IEEE International Conference on Data Engineering, 1988.

[3] Bernstein, P. and D.M. Chiu, "Using semi-joins to Solve relational queries," *JACM*, January 1981.

[4] Bernstein, P., N. Goodman, E. Wong, C. Reeve, J. Rothnie, "Query processing in a system for distributed databases (SDD-1)," *ACM Transactions on Database Systems,* December 1981.

[5] Breitbart, Y., P. Olson and G. Thompson, "Database integration in a distributed heterogeneous database system," Proc. IEEE International Conference on Data Engineering, 1986.

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

[7] Chen, A.L.P., "A localized approach to distributed query processing," to appear in Proc. EDBT 90 (Extending Data Base Technology).

[8] Chen, A.L.P., D. Brill, M. Templeton and C. Yu, "Distributed query processing in a multiple database system," *IEEE Journal on Selected Areas in Communications*, special issue on Databases in Communications Systems, Apr. 1989.

[9] Chen, A.L.P. and V.O.K. Li, "Optimizing star queries in a distributed database system," Proc. VLDB, 1984.

[10] Chen, A.L.P. and V.O.K. Li, "Improvement algorithms for semijoin query processing programs in distributed database systems," *IEEE Transactions on Computers,* November 1984.

[11] Chen, A.L.P. and V.O.K. Li, "An optimal algorithm for processing distributed star queries," *IEEE Transactions on Software Engineering,* October 1985.

[12] Chiu, D.M., P. Bernstein, Y.C. Ho, "Optimizing chain queries in a distributed database system," *SIAM J. Comput.,* February 1984.

[13] Codd, E.F., "Extending the database relational model to capture more meaning," *ACM TODS,* December 1979.

[14] Czejdo, B., M. Rusinkiewicz and D. Embley, "An approach to schema integration and query formulation in federated database systems," Proc. IEEE International Conference on Data Engineering, 1987.

[15] Date, C.J., "Null values in database management," Proc. Second British National Conference on Databases, 1982.

[16] Date, C.J., "The outer join," Proc. Second International Conference on Databases, 1983.

[17] Dayal, U., "Query processing in a multidatabase system," *Query Processing in Database Systems*, Kim, Reiner and Batory (editors), 1985.

[18] Dayal, U. and H. Hwang, "View definition and generalization for database integration in multibase: a system for heterogeneous distributed databases," *IEEE Trans. Softw. Eng.*, Nov. 1984.

[19] Deen, S., R. Amin and M. Taylor, "Data integration in distributed databases," *IEEE Trans. Softw. Eng.*, Jul. 1987.

[20] Gamal-Eldin, M., G. Thomas and R. Elmasri, "Integrating relational databases with support for updates," IEEE Proc. International Symposium on Databases for Parallel and Distributed Systems, 1988.

[21] Hammer, M. and S. Zdonik, "Knowledge-based query processing," Proc. VLDB, 1980.

[22] Hwang, H., U. Dayal and M.G. Gouda, "Using semiouterjoins to process queries in multidatabase systems," Proc. ACM PODS, 1984.

[23] King, J., *Query Optimization by Semantic Reasoning,* UMI Research Press, 1984.

[24] Shenoy, S.T. and Z.M. Ozsoyoglu, "A system for semantic query optimization," Proc. ACM SIGMOD, 1987.

[25] Smith, J. et al., "Multibase - Integrating heterogeneous distributed database systems," Proc. AFIPS NCC, 1981.

[26] Smith, J. and D.C.P. Smith, "Database abstractions: aggregation and generalization," *ACM TODS*, 1977.

[27] Stonebraker, M., "Implementation of integrity constraints and views by query modification," Proc. ACM SIGMOD, 1975.

[28] Templeton, M., D. Brill, A.L.P. Chen, S. Dao, E. Lund, R. MacGregor, P. Ward, "Mermaid - a front-end to distributed heterogeneous databases," *Proceedings of the IEEE*, May, 1987.

[29] Yu, C.T., C.C. Chang, M. Templeton, D. Brill, E. Lund, "On the design of a query processing strategy in a distributed database environment," Proc. of ACM SIGMOD, 1983.

[30] Yu, C.T., C.C. Chang, M. Templeton, D. Brill, E. Lund, "Query processing in a fragmented relational distributed system: MERMAID," *IEEE Transactions on Software Engineering*, August 1985.

[31] Yu, C.T., K. Guh, D. Brill, A.L.P. Chen, "Partition strategy for distributed query processing in fast local networks," *IEEE Trans. on Software Engineering*, June 1989.

[32] Yu, C., K. Guh and A.L.P. Chen, "An integrated algorithm for distributed query processing," Proc. IFIP Conference on Distributed Processing, 1987.

[33] Yu, C.T., L. Lilien, K. Guh, M. Templeton, D. Brill, A.L.P. Chen, "Adaptive techniques for distributed query optimization," Proc. IEEE International conference on Data Engineering, 1986.