# Query Processing Strategies in Heterogeneous and Distributed Multidatabase Systems

S. Mohsen Sedighi Janusz R. Getta
Department of Computer Science
University of Wollongong
Northfields Ave.,
Wollongong, NSW 2522
Australia
E-mail: [jrg, moh]@cs.uow.edu.au

June 11, 1996

**Abstract**

This paper addresses a problem of efficient query processing in heterogeneous and distributed multidatabase systems based on a global data model architecture. A query submitted to a global view of the component database systems is decomposed into a collection of subqueries. Selection of the best strategy for processing of subqueries has an important impact on the response time. The paper identifies two generic approaches to submission and computation of subqueries and proposes two algorithms that provide a near optimal solution. Moreover a number of optimization techniques are proposed for integration of the partial results into the final answer.

# Contents

**1 Introduction** **3**

**2 Basic concepts** **5**

**3 Query computation strategies** **7**

**4 Query postprocessing** **10**

**5 Summary** **12**

**A Appendix** **15**

# List of Figures

1 A labelled syntax tree of an expression $(q_1 \cap q_2) \bowtie (q_3 - q_4)$. . . . . 6
2 A labelled syntax tree of an expression $(q_1 \cup q_2) \bowtie (q_3 - (q_4 - q_5))$. . . . . 7
3 Labelling of a syntax tree to find reduction of the results of $q_j$. . . . . 7
4 Labelling of a syntax tree that leads to reduction of $q_4$. . . . . 8
5 A skeleton of syntax tree with the common subtrees. . . . . 11

# List of Tables

1 Labelling for binary operations. . . . . 6
2 Labelling for unary operations. . . . . 7
3 Reduction table for join operation ($k = x \cap y$). . . . . 8
4 Hidden dependencies for join operation ($k = x \cap y$). . . . . 11
5 Reductions for intersection operation ($k = x \cap y$). . . . . 15
6 Reductions for union operation ($k = x \cap y$). . . . . 15
7 Reductions for difference operation when a reduced subquery is on the right hand side of difference operation ($k = x \cap y$). . . . . 15
8 Reductions for difference operation when a reduced subquery is on the left hand side of difference operation ($k = x \cap y$). . . . . 15

# 1 Introduction

Optimization of query processing in heterogeneous and distributed multidatabase systems [SheL90] requires different techniques than the classical methods of query processing in distributed homogeneous database systems [BGWR81, ApHY83, CerP85, ChuH82, YuLi90]. There are several reasons why the techniques developed for homogeneous distributed database systems cannot be directly applied to heterogeneous distributed multidatabase systems [RePR89, SchC92, BreT85]. One of them is a high level of autonomy preserved by the heterogeneous components of multidatabase database systems [GarK88, Corn88]. Security requirements of the individual database sites contradict more intelligent query processing based on collaboration of the component database sites. A high level of autonomy enforces a simple *receive-compute-respond* pattern of query processing in heterogeneous multidatabase systems. The other problems include software, syntax and semantic heterogeneities [SheL90] of the component database systems. Software heterogeneity means that different database sites are managed by different implementations of database management systems. Syntax heterogeneity refers to different logical and physical models of the component databases. Finally, semantic heterogeneity means that the same or similar database structures may have different meanings in different databases. Software and syntax heterogeneities require translation of the queries and the results of their computation from one formalism into the other. Such processes; especially in the case of data translation, can be extremely time consuming. Semantic heterogeneities affect the use of replicated data because of the problems with recognition which replicas stored in different databases have exactly the same semantics. Yet another source of troubles is rather anarchistic model of control over the database components of heterogeneous multidatabase system. Actually, none of the participating database sites has global control over the other databases sites. As the result a database site responsible for global query computation has no impact on the priority of query processing in the other sites. Therefore, it is impossible to enforce the faster processing of subqueries which are critical for the total query processing time. All these problems do not exist in homogeneous and distributed systems.

This paper proposes two algorithms for selection of the best query processing strategy in heterogeneous, distributed multidatabase systems. In our model a multidatabase system consists of several independent and heterogeneous database systems, henceforth known as *component databases*. Each component of the system is located at a separate site on the network. A user obtains a unified view of all components through a common database model. A common view is implemented by additional software running on the top of a local DBMS and commonly called as *middleware*. Middleware is responsible for decomposition of user queries into collections of subqueries, translation of subqueries, coordination of query processing, communication with other sites and translation of data into a common format. A middleware has no impact on autonomy of the component database systems. A message passing system implements communication between the component databases. It is assumed that a message contains either a query submitted for execution or the results of query processing.

In the initial phase of query processing, a middleware decomposes a query Q submitted by the end-user into a set of subqueries $\{q_1, \ldots, q_k\}$ such that computation of each one of them requires access to only one component database [Daya85]. Next, each subquery is translated into the query language of a corresponding component database system before it is submitted for execution.

The subqueries are submitted to and computed in the components of multidatabase systems accordingly to a certain strategy. Query computation strategy has an important impact on entire performance of query processing. We identify two generic query computation strategies. The first one attempts to use the partial results which have been obtained from the processing of one subquery to reduce size of the results obtained from the processing of the other subqueries. According to this strategy a subquery $q_i$can be submitted for execution only if the results from the earlier submitted subquery $q_{i-1}$ are available. This is why it is called as *sequential compu-*

*tation strategy.* As an example consider a multidatabase system that integrates the component databases under a common view of relational database model. Moreover, let the system provides a view of the relational tables r and s implemented in the separate component databases $d_1$ and $d_2$. Because of heterogeneities, implementation of r and s in the component databases does not need to be relational and it does not need to be the same. Let x be a set of the attributes common to r and s. A query r $\bowtie$ s submitted to a common view of a multidatabase system is decomposed into two subqueries $q_{d_1}(r)$ and $q_{d_2}(s)$ expressed in the query languages of the respective database sites $d_1$ and $d_2$. According to a sequential submission strategy the system first submits $q_{d_1}(r)$ to $d_1$. The results obtained from $d_1$ are translated into a relational table r. In homogeneous distributed systems $\pi_x(r)$ can be transmitted to site $d_2$ to compute s $\ltimes$ $\pi_x(r)$. Semijoin reduces amount of data transmitted from $d_2$. Unfortunately, such approach cannot be applied in a heterogeneous multidatabase system because each one of the component database systems accepts queries but not data. However there exits a solution to this problem. If $\pi_x(r)$ is not too large, then it is translated into a preposition formula $\phi(\pi_x(r))$. Each tuple from $\pi_x(r)$ is translated into a conjunction of atomic terms *attribute=value*. Entire formula is a disjunction of the conjunctions obtained from the translation of individual tuples. If table $\pi_x(r)$ is too large then it is partitioned into the disjoint tables $\pi_x(r_1)$, ..., $\pi_x(r_k)$ such that r = $r_1 \cup \ldots \cup r_k$. Then, each one of the tables is separately translated into $\phi(\pi_x(r_1))$, ..., $\phi(\pi_x(r_k))$. Finally, the second subquery is modified into $q_{d_2}(\sigma_{\phi(\pi_x(r))}(s))$ before its submission. The objective of this approach is to reduce amount of data transmitted from site $d_2$. Even in such simple example selection of the appropriate order of subquery computation has an important impact on the global performance. If table r is significantly larger than s and computation of $q_{d_1}(\sigma_{\phi(\pi_x(s))}(r))$ strongly reduces size of the result then it is better to compute $q_{d_1}(s)$ first. Therefore an order in which the subqueries are computed has an important impact on the overall performance of the system.

A sequential query computation strategy does not provide the best results in every possible case. For instance, when the amount of time spent on processing of $q_{d_1}(r)$ and $q_{d_2}(\sigma_{\phi(\pi_x(r))}(s))$ dominates over the transmission time then sequential computation of subqueries is not the best solution. In such a case it is better to simultaneously submit $q_{d_1}(r)$ and $q_{d_2}(s)$ for their "parallel" processing in $d_1$ and $d_2$. A strategy where all subqueries are simultaneously submitted and computed in parallel the component databases is known as *parallel computation strategy* [YuLi90]. Such an approach is beneficial if parallel computation of subqueries in the component database saves more time than the hypothetical savings on transmission time when sequential strategy is applied.

Parallel computation of subqueries seems to be somewhat more flexible in heterogeneous distributed multidatabase systems because, in addition to heterogeneity and autonomy of component databases, not all of the required database operations exist in all the component databases. As usual the best solution is a hybrid strategy where some of subqueries are submitted and computed sequentially yet the others are submitted and computed in parallel. Identification of the best query computation strategy depends on the factors such as the computational complexity of the individual subqueries, the size of the respective results, the efficiency of query processing in the component databases and the network parameters. Some of these problems have been already addressed. For instance, the works [DuKS92, ZhuL94, HaSN96] deal with the problem of subquery costs estimation within component databases. The optimization of query processing in multidatabase systems has been discussed in [LuSh92, LuOG93, LimS93].

Our objectives are as follows. Given a query Q submitted to a multidatabase system and its decomposition into a set of subqueries $\{q_1, \ldots, q_k\}$. The problem is to find a strategy for computation of subqueries in the component databases that minimize the total computation time of query Q. We split a problem into two subproblems. First we look for the best sequential strategy, i.e. for the linear order of subqueries $<q_{i_1}, \ldots, q_{i_k}>$ that minimize a given cost function. Then we try to find all subsequences of subqueries $<q_{j_1}, \ldots, q_{j_n}>$ n<=k within $<q_{i_1}, \ldots, q_{i_k}>$ such that their parallel computation further minimizes the cost function.

To find the best linear order for the computation of subqueries we pick one of them, e.g. $q_i$ and we assume that subquery $q_i$ was computed first and its results are available at our site. Then we find what reductions of transmission size of the results from $q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_k$ can be done using the results of $q_i$. The computations are repeated for each $q_i$, i=1, ..., k and a subquery which provides the highest reductions is selected as the first in the linear order. Such procedure is repeated for the other subqueries until all of them are ordered. The next step is to find the subsequences of queries such that their parallel computations reduce the response time. Each time a group of subqueries $<q_{j_1}, \ldots, q_{j_n}>$ is identified, it is replaced with an identifier of a single subquery $q_{j_1, \ldots, j_n}$ and identification of the next group is repeated.

An interesting problem is detection of an impact the results of $q_i$ have on reduction of the results of $q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_k$. The following example provides more intuitions related to our solution. Given a query Q and its decomposition into subqueries r, s, t such that Q = r - (s - t). Let us assume that subquery r is computed first and v = (s - t). Then, the results of r can be used to reduce v to $\sigma_{\phi(r)}(v)$ where $\phi(r)$ is an operator that transforms the results of r into a prepositional formula. Because selection is distributive over difference then s and t can be reduced to $\sigma_{\phi(r)}(s)$ and $\sigma_{\phi(r)}(t)$. On the other hand assume that subquery t is computed first. Then, its results can be used to reduce s into $\sigma_{not\phi(t)}(s)$. In this case the results of t have no impact on reduction of r because t contains only some of the tuples that belongs to the result of r - (s - t). In the third case, i.e. when subquery s is computed first its results can be used to reduce t into $\sigma_{\phi(s)}(t)$, but there is no impact on r. Finally, estimation of the reduction sizes determines which subquery should be computed first.

Another kind of optimization can be done when the subqueries are computed in parallel. This kind of optimization is done in a query postprocessing phase, i.e. when the partial results are integrated into the final answer. It may happen in the parallel strategy that the system should wait because some of the partial results are not available on time. A question arise what part of integration can be done if only some of the partial results are available at the central site. For instance consider the example given above and assume that subqueries $q_1$, $q_2$, $q_3$ are computed in parallel. Moreover, let us assume that in a certain moment of time only $q_1$ and $q_2$ are available for further processing. . Is it possible to compute a part of expression $q_1^*$ - ($q_2^*$ - $q_3$); the arguments available are superscripted with *, while the system is waiting for a partial result $q_3$ ? In this case a simple algebraic transformation solves the problem. An expression $q_1^*$ - ($q_2^*$ - $q_3$) is equivalent to an expression ($q_1^*$ - $q_2^*$) $\cup$ ($q_1^* \cap q_3$). Therefore, it is possible to compute a part of the final answer ($q_1^*$ - $q_2^*$) without the third argument.

The paper is structured as follows. The next section contains the definitions of the basic concepts and a general method for identification of subquery reductions. Section 3 discusses selection of the best sequential and hybrid (sequential and parallel) strategies of query computation. Section 4 reviews the techniques of postquery optimization of subqueries computed in parallel. Section 5 concludes the paper.

## 2 Basic concepts

We consider a model of multidatabase system that provides its users with a relational view of the component databases. The users express the queries in a high level query language of relational database model. The system translates a query into a relational algebra expression where each one of its arguments requires access to a maximum of one of the component databases. Then, the system generates subqueries $\{q_1, \ldots, q_k\}$ and translates them into the query languages of the corresponding component databases $\{d_1(q_1), \ldots, d_k(q_k)\}$. A relational algebra expression $e(q_1, \ldots, q_k)$ determines how the final answer should be constructed from the partial results. We call it as *data integration expression.*

Let $\pi$, $\sigma$, $\bowtie$, $\ltimes$, $\cap$, $\cup$, "-" denote the standard operations of relational algebra. A *syntax tree* of a data integration expression is a binary tree where the internal nodes are labelled with the operations of relational algebra and the leaves are labelled with the arguments.

Selection of the best sequential query computation strategy requires estimation of the reductions that can be done on $q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_k$ when the results of $q_i$ are available. We solve this problem by labelling a syntax tree of data integration expression. All edges on the paths from the subscript leaf nodes to a root node are labelled to represent the flow of data from one operation to another. A label consists of a qualifier and argument name for which a flow of data is considered. There are five qualifiers: A, A+, E, E+, N+.

Let $\alpha$ , $\beta$ be the operation nodes in a syntax tree such that result of $\alpha$ becomes an argument of $\beta$. Moreover, let subquery q be an argument of $\alpha$. An edge from $\alpha$ to $\beta$ obtains a label A(q) if operation $\alpha$ returns all (A) and no other tuples from the result of q. An edge from $\alpha$ to $\beta$ obtains a label A(q)+ if a result of operation $\alpha$ returns all tuples from the result of q plus some other tuples. An edge from $\alpha$ to $\beta$ is labelled by E(q) if in the result of $\alpha$ there exist some (E) tuples from q and there is no other tuples. An edge from $\alpha$ to $\beta$ is labelled by E(q)+ if operation $\alpha$ returns some tuples from the result of q plus some other tuples. Finally an edge from $\alpha$ to $\beta$ obtains a label N(q)+ if operation $\alpha$ returns no (N) tuples from q plus some other tuples. As an example consider a data integration expression $(q_1 \cap q_2) \bowtie (q_3 - q_4)$ whose labelled syntax tree is given in Figure 1. An edge between the nodes ($\cap$) and ($\bowtie$) is labelled by $E(q_1)$. If a set of all

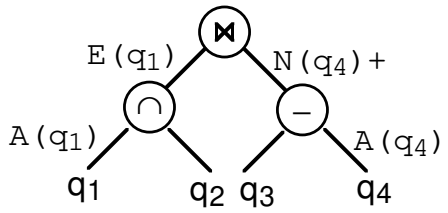


Figure 1: A labelled syntax tree of an expression $(q_1 \cap q_2) \bowtie (q_3 - q_4)$.

tuples obtained from computation of $q_1$ ($A(q_1)$) is intersected with the other set of tuples then the result contains some of the tuples from $q_1$ ($E(q_1)$) and no other tuples. An edge between the nodes (-) and ($\bowtie$) is labelled by $N(q_4)+$. If a set of all tuples obtained from $q_4$ ($A(q_4)$) is subtracted from a set of tuples obtained from $q_3$ then the result contains no tuples from $q_4$ ($N(q_4)+$) and some other tuples from $q_3$.

Labelling of the edges on the higher levels of syntax tree is described by Table 1 and Table 2.

| *label\operation* | *join* | *intersection* | *l-difference* | *r-difference* | *union* |
|---|---|---|---|---|---|
| A(q) | E(q) | E(q) | E(q) | N(q)+ | A(q)+ |
| A(q)+ | E(q)+ | E(q)+ | E(q)+ | N(q)+ | A(q)+ |
| N(q)+ | N(q)+ | N(q)+ | N(q)+ | E(q)+ | E(q)+ |
| E(q) | E(q) | E(q) | E(q) | E(q)+ | E(q)+ |
| E(q)+ | E(q)+ | E(q)+ | E(q)+ | E(q)+ | E(q)+ |

Table 1: Labelling for binary operations.

For instance, the leftmost column in Table 1 determines the labelling in all cases when a node represents join operation and one of its argument edges is labelled either by A(q), or A(q)+, or E(q), or E(q)+, or N(q)+. Then, the result edge of a join node should be labelled respectively by either E(q), or E(q)+, or N(q)+, or E(q), or E(q)+. Because difference operation is not commutative we have to separately consider the cases when an argument edge represents the left or right argument. Figure 2 provides a labelled syntax tree for an expression $(q_1 \cup q_2) \bowtie (q_3 - (q_4 - q_5))$. An edge between the nodes ($\bowtie$) and (-) is labelled by $E(q_5)+$ because an argument edge of a node (-) is labelled by $N(q_5)+$ and in Table 1 intersection of N(q)+ row and right-difference column contains a label E(q)+. If a node represents projection operation then a qualifier of edge label is indexed by a set of projected attributes. For instance $E_x(q)+$ means

| *label\operation* | *selection* | *projection* |
|---|---|---|
| A(q) | E(q) | $A_x$(q) |
| A(q)+ | E(q)+ | $A_x$(q)+ |
| N(q)+ | N(q)+ | $N_x$(q)+ |
| E(q) | E(q) | $E_x$(q) |
| E(q)+ | E(q)+ | $E_x$(q)+ |

Table 2: Labelling for unary operations.

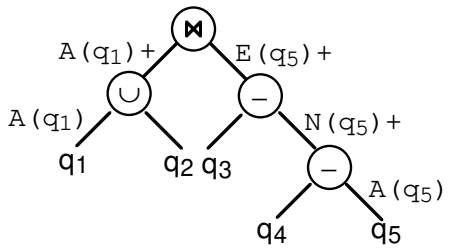


Figure 2: A labelled syntax tree of an expression $(q_1 \cup q_2) \bowtie (q_3 - (q_4 - q_5))$.

that result of operation contains some of x-values of the tuples from r plus some other x-values. Note that the binary operations represented in Table 1 do not change an index of qualifier.

In order to find an impact of the results obtained after computation of subquery $q_i$ on size reduction of the results obtained from subquery $q_j$ we have to label the paths from both arguments to a node which is a root of their common subtree. Then, depending on operation $\Psi$ in the root node and the labels $L_i$ and $L_j$ of the topmost edges (see Figure 3, it is possible to find modification of a subquery $q_j$ to reduce transmission size (see Table 3 in the following and tables 5, 6, 7, 8 in Appendix A). For instance labelling of a syntax tree of data integration expression $(\pi_x(q_1 - q_2) \cap q_3) \cap (\sigma_\phi(q_4) \cup q_5)$ is given in Figure 4. Intersection of a column marked by $E(q_j)+$ and a row marked by $E(q_i)$ in Table 4 suggests that $q_4$ can be replaced with $\sigma_{\phi(\pi_x(q_1))}(q_4)$. Whenever an entry in the tables 3, 4, 5, 6 is equal to NIL then reduction cannot be done.

## 3 Query computation strategies

Initially we shall concentrate on selection of the best linear order for sequential computation of subqueries. A naive solution to this problem would be to compute the total query processing costs for all possible linear orders. Unfortunately, complexity of such solution is beyond NP-completeness [GarJ79] because for k subqueries we have to consider k! cases. We shall apply

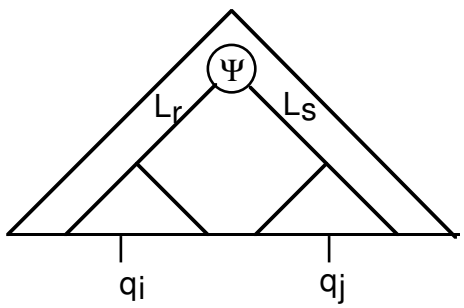


Figure 3: Labelling of a syntax tree to find reduction of the results of $q_j$.

| $\bowtie$ | $A(q_j)$ | $A(q_j)+$ | $N_y(q_j)+$ | $E_y(q_j)$ | $E_y(q_j)+$ |
|---|---|---|---|---|---|
| $A(q_i)$ | $\sigma_{\phi(q_i)}(q_j)$ | $\sigma_{\phi(q_i)}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ |
| $A(q_i)+$ | NIL | NIL | NIL | NIL | NIL |
| $N_x(q_i)+$ | NIL | NIL | NIL | NIL | NIL |
| $E_x(q_i)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ |
| $E_x(q_i)+$ | NIL | NIL | NIL | NIL | NIL |

Table 3: Reduction table for join operation (k = x ∩ y).

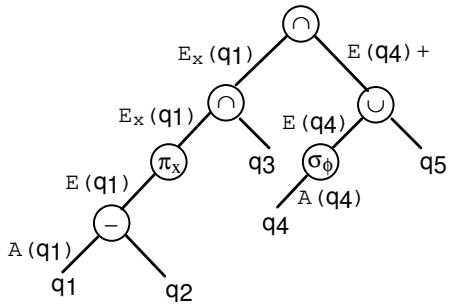


Figure 4: Labelling of a syntax tree that leads to reduction of $q_4$.

the greedy method of searching for near optimal solution [HorS78]. The idea is to find what are the costs of computation of subquery $q_i$ for all i=1, ..., k. Next, we label a syntax tree of data integration expression to find all subqueries $q_j^{(i)}$ for j=1,...,k, i≠j, where $q_j^{(i)}$ denotes $q_j$ modified by the results of $q_i$. Analysis of all $q_j^{(i)}$ provides estimation of their costs. A pair $<q_i, q_j^{(i)}>$ with a minimal $cost(q_i) + cost(q_j^{(i)})$ is accepted as the first two elements of the order. Next, we find all subqueries $q_n^{(i,j)}$ for all n=1,...,k, k≠j, k≠i, and their costs. Subquery $q_n^{(i,j)}$ with the minimum cost is accepted as the next element of the sequence, and so on.

The costs of $q_i$ processing at a component database site depends on query transmission time to a component database; $\tau(q_i)$, query processing time at a component database; $\tau_{process}(q_i)$, results transmission time;$\tau_{result}(q_i)$, data conversion time; $\tau_{convert}(q_i)$ and a time which is needed to integrate $q_i$ with the results of the other subqueries $\tau_{integrate}(q_i)$.

$$cost(q_i) = \tau(q_i) + \tau_{process}(q_i) + \tau_{result}(q_i) + \tau_{convert}(q_i) + \tau_{integrate}(q_i, e) \quad (1)$$

Query transmission time and query processing time can be estimated from the size and complexity of the query. Time which is needed to transmit the results and data conversion time depends on the amount of data obtained from processing of $q_i$ and it can be estimated from the previous results. Data integration time is computed using the syntax tree of data integration expression and the amounts of data obtained from processing of $q_i$, i=1, ..., k. The total cost of $q_j^{(i)}$ is computed in a similar way.

$$cost(q_j^{(i)}) = \tau(q_j^{(i)}) + \tau_{process}(q_j^{(i)}) + \tau_{result}(q_j^{(i)}) + \tau_{convert}(q_j^{(i)}) + \tau_{integrate}(q_j^{(i)}, e) \quad (2)$$

If the results of subquery $q_i$ are too big to be translated into a preposition formula we have to split them into $q_{i1}$, ..., $q_{in}$. Then,

$$cost(q_j^{(i)}) = \Sigma_{m=1,...,n}(\tau(q_j^{(im)}) + \tau_{process}(q_j^{(im)}) + \tau_{result}(q_j^{(im)}) + \tau_{convert}(q_j^{(im)}))+$$

$$cost(q_{i1} \cup \ldots \cup q_{in}) + \tau_{integrate}(q_j^{(i)}, e) \quad (3)$$

The total cost of query processing for a pair $<q_i, q_j>$ is the summation of (1) and (2). The most important factor for computation of subquery processing cost is estimation of the size of

the results obtained from the component databases. Amount of data obtained as the result of subquery processing at a component site determines data transmission, data conversion, and data integration time. This problem has been investigated in [DuKS92, ZhuL94, HaSN96] and we shall skip it in the rest of the present work.

An algorithm given below constructs a sequence s of subqueries that determines a sequential query processing strategy which is a near optimal solution.

**Algorithm 3.1**

*Let the initial sequence s be empty.*
*for all i = 1, ..., k*

*compute cost($q_i$);*
*for all j =1, ..., k j≠i*
*compute cost($q_j^{(i)}$);*
*find <$q_i$, $g_j^{(i)}$> such that cost(<$q_i$, $g_j^{(i)}$>) is minimal;*
*s = <$q_i$, $q_j^{(i)}$>;*
*repeat*

*for all n = 1, ..., k and n ≠ any of indices of subqueries in s*
*compute cost($q_n^{(\text{indices of queries in } s)}$);*
*find subquery $q_n^{(\text{indices of queries in } s)}$ such that its cost is minimal and append it at the end of s;*
*until all subqueries have been appended to s;*

Algorithm 3.1 can be simplified if we assume that the changes to cost($q_i^{(j)}$) for i,j=1,...,k i≠j are insignificant during computation of a sequence s. Then there is no need to recompute cost($q_n^{(\text{indices of queries in } s)}$) for n=1,...,k and n ≠ any of indices of subqueries in s, each time a new element is appended to s.

If cost(<$q_i$, $q_j^{(i)}$>) computed by the Algorithm 3.1 is greater than max(cost($q_i$), cost($q_j$)) then execution of the subqueries $q_i$and $q_j$ in parallel provides better results than the sequential one. An algorithm given below detects all subsequences of the queries in a sequence constructed by Algorithm 3.1 whose parallel computation provides better performance.

**Algorithm 3.2**

*Let s = <$q_1$, ..., $q_k$>, let i = 1;*
*repeat*

*Find the longest subsequence of queries $q_i$, ..., $q_n$, n≤k in s such that*
*cost(<$q_i$, ..., $q_n$>) > max(cost($q_i$), ..., cost($q_n$));*
*Replace $q_i$, ..., $q_n$ with ($q_i$|...|$q_n$) in s to denote that $q_i$, ..., $q_n$ should be computed in parallel;*

*i = n+1;*
*until i > k;*

The first pass has been completed. Next, construct a new sequence of subqueries s' replacing each ($q_i$|...|$q_n$) in s with a query symbol $q_{i...n}$, and assuming that cost($q_{i...n}$) = max(cost($q_i$), ..., cost($q_n$)). Repeat the algorithm from the beginning of a new sequence s'. Continue, until no new parallel computations can be discovered. At the end, replace each symbol $q_{i...n}$ with the respective sequence ($q_i$|...|$q_n$).

As the result of Algorithm 3.2 we obtain a string of subqueries where each sequence of subquery symbols separated by a comma denotes sequential execution and each sequence of subqueries within the brackets and separated by a vertical bar denotes sequential execution.

# 4 Query postprocessing

So far we have concentrated on selection of the best subquery computation strategy. When parallel query computation strategy is applied it is possible to look for additional improvements in system performance in a phase of query postprocessing. The objective of query postprocessing is integration of the partial results obtained from the component database sites into the final answer.

Although the subqueries are submitted simultaneously and computed in parallel their results do not return to a submission site at exactly the same moment of time. The delay is due to the high level of autonomy of the component databases, low transmission speed, poor hardware of a system that runs a component database, or the high computational complexity of a subquery. It is possible to commence a gradual integration of the partial results even when not all of them are available. We show how to utilize the time the system spends waiting for the partial results on the preparation of those answers already available with a view to more effective integration with other results.

Preparation of the partial results is possible on both logical and physical levels. On the physical level pre-hashing of relational tables speeds up computation of join operations. Indexing and clustering of the partial results are the other options. Preparation of the partial results on the logical level resembles classical syntax-based query optimization [Ullm89]. A data integration expression is transformed into an equivalent form so that it can be computed faster when the results of all subqueries are available. We shall concentrate on the preparation of the partial results on the logical level. The problem is as follows. Given a data integration expression $e(q_1, \ldots, q_k)$ and the results of computation of some of the queries $q_1, \ldots, q_k$. Find what operations can be performed using the available data to obtain a part of the final answer (whenever it is possible), how to reduce size of the available results, and how to prepare them for further processing.

The objective of the first group of optimizations is to transform data integration expression $e(q_1, \ldots, q_k)$ into an equivalent form such that the arguments available for further computations are bound by the common operations and the total number of operations does not increase. This kind of transformations relies on application of the associative and distributive laws of relational algebra. For instance, ${q_1}^* \bowtie q_2 \bowtie \ldots \bowtie {q_k}^*$ (assume that ${q_i}^*$ denotes the results of $q_i$ which are already available) becomes ${q_1}^* \bowtie {q_k}^* \bowtie q_2 \bowtie \ldots \bowtie q_{k-1}$.

The next objective is to distribute binary operations over union operations to obtain an expression which is a union of subexpressions such that each one of them contains no union operation and one of them can be completely computed using the available arguments. Distribution of the binary operations is always possible except the case when a union subexpression is on the right hand side of difference operation e.g. ${q_1}^* - ({q_2}^* \cup q_3)$. In this case we apply a transformation rule that replaces union with a difference, e.g. $({q_1}^* - {q_2}^*) - q_3$. Such transformations are reasonable if the leftmost subexpression can be immediately computed. In general, distribution of binary operations over union increases a total number of operations in a data integration expression. Therefore we should consider it beneficial only if one of the union subexpressions can be immediately computed and a part of the final result may be delivered to the user even though not all of the arguments of the data integration expression are available. For example transformation of the expression $( {q_1}^* \cup q_2) \bowtie {q_3}^*$ into $({q_1}^* \bowtie {q_3}^*) \cup (q_2 \bowtie {q_3}^*)$ is recommended because $({q_1}^* \bowtie {q_3}^*)$ provides a part of the final answer.

The third kind of transformation reshuffles the available results of subquery processing by replacement of the difference operations with the intersection and complement of the argument and by the application of the distributive laws of set algebra. This transformation is limited to only a subclass the data integration expression. As an example consider the expression $(q_1 - {q_2}^*) - ({q_3}^* \cup q_4)$. After the elimination of any difference it is equivalent to $(q_1 \cap \neg({q_2}^*)) \cap \neg({q_3}^* \cup q_4)$ where $\neg$ denotes the set complement. Finally, using De Morgan's laws for sets and eliminating the complement operations we obtain $( q_1 - ({q_2}^* \cup {q_3}^*)) - q_4$. The transformation

allows the immediate computation of ($q_2^* \cup q_3^*$) and thus eliminates one operation.

The objective of the next group of transformations is to reduce the size of the arguments of the data integration expression through the identification of *hidden dependencies*. A hidden dependence between two not adjacent arguments of a data integration expression exists when it is possible to construct an expression that involves both arguments, such that its result is included in one of them and it can replace the argument without affecting the computation of the data integration expression in any possible case.

For instance consider a data integration expression $(\pi_x(q_1^* - q_2) \cap q_3) \cap (\sigma_\phi(q_4^*) \cup q_5)$. There exists a hidden dependence between $q_1$ and $q_4$ because $q_4$ can be replaced with the result of $q_4 \ltimes \pi_x(q_1)$ and such replacement does not affect the result of computation of the data expression for arbitrary values of $q_2$, $q_3$, and $q_5$. Intuitive justification of this conclusion is as follows. If a result of $q_4$ contains a tuple t such that there is no tuple in $q_1$ that has the same x-values as t then it will never be a member of the final result and thus it can be removed from the result of $q_4$. This is true because the difference and intersection operations in an $\pi_x(q_1^* - q_2) \cap q_3$ do not add any new tuples to $\pi_x(q_1)$.

The hidden dependencies can be detected by labelling the syntax tree of the data integration expression in the same way as it has been explained in Section 2. A table given below includes all reductions which are possible for the given combinations of labels at the topmost edges when join operation is in the root of subtree common to $q_i$ and $q_j$. The tables for the other operation can be constructed in the similar way. An optimization technique based on marking the tuples

| $\bowtie$ | $A(q_j)$ | $A(q_j)+$ | $N_y(q_j)+$ | $E_y(q_j)$ | $E_y(q_j)+$ |
|---|---|---|---|---|---|
| $A(q_i)$ | compute $q_i \bowtie q_j$ | $q_j = q_j \ltimes q_i$ | $q_j = q_j \ltimes \pi_k(q_i)$<br>$q_i = q_i \ltimes \pi_k(q_j)$ | $q_j = q_j \ltimes \pi_k(q_i)$ | $q_j = q_j \ltimes \pi_k(q_i)$ |
| $A(q_i)+$ | $q_i = q_i \ltimes q_j$ | NIL | NIL | $q_i = q_i \ltimes \pi_k(q_j)$ | NIL |
| $N_x(q_i)+$ | $q_i = q_i \ltimes \pi_k(q_j)$ | NIL | NIL | $q_i = q_i \ltimes \pi_k(q_j)$ | NIL |
| $E_x(q_i)$ | $q_i = q_i \ltimes \pi_k(q_j)$ | $q_j = q_j \ltimes \pi_k(q_i)$ | $q_j = q_j \ltimes \pi_k(q_i)$ | $q_i = q_i \ltimes \pi_k(q_j)$ | $q_j = q_j \ltimes \pi_k(q_i)$ |
| | $q_j = q_j \ltimes \pi_k(q_i)$ | | | $q_j = q_j \ltimes \pi_k(q_i)$ | |
| $E_x(q_i)+$ | $q_i = q_i \ltimes \pi_k(q_j)$ | NIL | NIL | $q_i = q_i \ltimes \pi_k(q_j)$ | NIL |

Table 4: Hidden dependencies for join operation (k = x ∩ y).

of available arguments is important when a data integration expression has one of two or more identical subexpressions such that there exists a hidden dependence in one of them. For instance consider a skeleton of data integration expression given in Figure 5. A subtree T is common for

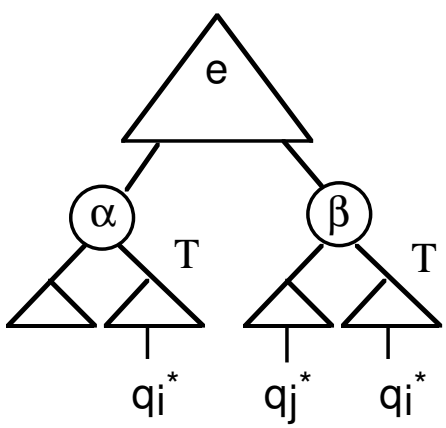


Figure 5: A skeleton of syntax tree with the common subtrees.

the operations $\alpha$ and $\beta$ but only in $\beta$ case there exists a hidden dependence between the results of $q_i$ and $q_j$. If we assume that dependence between $q_i$ and $q_j$ allows the replacement of $q_i$ with

f($q_i$, $q_j$) then the right hand side subtree T becomes different from the left hand side subtree T. Unfortunately such optimization is not correct because it is no longer possible to compute T once. The solution to this problem is in the marking of the tuples in $q_i$ with (+) and (-) markers and clustering $q_i$ according to the markers possessed by the individual tuples. The tuples that belong to f($q_i$, $q_j$) are marked with (+) while the other tuples, i.e. $q_i$ - f($q_i$, $q_j$) are marked with (-). Moreover, the computation of the subexpression T should be done in such a way that it returns two results, one for evaluation of $\alpha$ and the other one for evaluation of $\beta$. The result of T which is later on used as an argument of $\alpha$ contains all tuples independent of their markers. On the other hand, every tuple in the result of T which becomes an argument of $\beta$ and such that it consists of a part or entire tuple marked with (-) should be discarded before computation of $\beta$. Such a tuple does not affect the result of $\beta$.

As an example consider a data integration expression ($q_1$* $\cup$ $q_2$) - ($q_3$* $\bowtie$ $q_4$). Labelling of the paths from the leaf nodes $q_1$* and $q_3$* to the root node (-) provides the labels A($q_1$)+ and E($q_3$). A hidden dependence between $q_1$ and $q_3$ does not exist. According to other optimization techniques the expression should $\beta$e transformed into a union of subexpressions ($q_1$* - ($q_3$* $\bowtie$ $q_4$)) $\cup$ ($q_2$ - ( $q_3$* $\bowtie$ $q_4$)). Labelling in the first subexpression indicates that $q_3$ can be replaced with the result of $q_3$ $\ltimes$ $q_1$. Unfortunately, the reduction of $q_3$ is not correct because it destroys common subexpressions. Therefore, $q_3$is marked and clustered into $q_3$' = $q_3$ $\ltimes$ $q_1$ and $q_3$" = $q_3$ - $q_3$'. When the result of $q_4$ is available then the entire result of $q_3$* $\bowtie$ $q_4$* is stored in order to compute the second subexpression while only $q_3$'* $\bowtie$ $q_4$ is kept for computation of the first subexpression.

In the following, we provide a skeleton of an algorithm for query postprocessing.

**Algorithm 4.1**

Let $q_1$, ..., $q_m$ be the subqueries computed in parallel in the component database systems. Let $R$ be a set of their results available for data integration. Initially $R = \emptyset$.

When a partial result $q_i$ arrives after time $t_i$ it is inserted into $R$. The system periodically inspects the set of partial results. If $R$ is not empty, i.e. it contains the partial results $q_i$, $q_{i+1}$, ..., $q_j$ then the following actions take place.

All operation that have all of their arguments available are immediately computed and the partial results of $q_i$, $q_{i+1}$, ..., $q_j$ are removed from R. The syntax tree is modified after the computations. The system inspects the set $R$ and if it is not empty the same actions are repeated for the new arrivals. At any moment of time the system is able to determine amount of time $t_a$ which is left to arrival of the next partial answer $q_{j+1}$.

The system identifies an appropriate syntax-based optimization techniques to reshuffle the arguments, to compute a part of the final answer, and to eliminate some of the operations. All optimizations that require less time than $t_a$ are performed. Then, the system inspects $R$ for new arrivals. If it is not empty the previous step is repeated. Otherwise the system continues optimization.

The system identifies the hidden dependencies and all possible reductions of the available results. All optimization that can be done before arrival of the next partial result are performed. If $R$ is still empty the system continues optimizations, otherwise it returns to the first step.

The system identifies all cases when reduction of the arguments conflicts with computation of the common subexpressions. In such cases the tuples from respective arguments are marked. If a set $R$ is still empty the system waits for new partial arguments as no other optimizations can be done. Otherwise the system returns to the first step.

# 5 Summary

This work discusses selection of the best query processing strategy in heterogeneous distributed database systems. A query submitted to a global view of a heterogeneous system is decomposed

into a collection of subqueries. Our goal is to find the best strategy for submission of subqueries to the component databases. We identify two generic strategies. According to the sequential strategy, subqueries are computed one by one and the results of the already processed subqueries are used to speed up processing of the others. In parallel strategy the system submits all subqueries simultaneously. Their execution in the component databases and transmission of the results are performed in parallel. There is a number of factors that have an impact on performance of query processing in such a model.. The sequential strategy takes under consideration the reduction of data transmission time while the parallel strategy attempts to minimize query computation time. In many cases the best solution is a hybrid strategy where some of the subqueries are processed sequentially while the others in parallel. We provide two algorithm that find a near optimal hybrid strategy. The first algorithm selects a near optimal sequential strategy. The second algorithm analyses its results to detect any parallelism that can minimize query processing time. Labelling of a syntax tree of data integration expression plays an important role for both algorithms. Analysis of the labels provides information about the reductions in data transmission time.

Parallel processing of subqueries allows for additional optimizations when integration of the results of subqueries is performed. We propose three optimization techniques to compute a data integration expression when its arguments arrive at a central site in the different moments of time. Like in the previous case, labelling of a syntax tree of data integration expression and analysis of the labels enables detection of the computations that can be done while the system is waiting for the missing arguments.

An interesting problem that is left open is more precise estimation of an impact that the results of already computed subqueries have on reduction of the other subqueries. It is intuitively obvious that smaller results of subquery $q_i$ have larger impact on reduction of the other results. A more quantitative analysis is needed. Another problem is an advanced query postprocessing strategy capable of adjusting itself dynamically to unexpected query processing delays in the component databases.

## References


[ApHY83] Apers P. M. G., Hevner A. R., Yao S. B. Optimization Algorithms for Distributed Queries, *IEEE Transactions on Software Engineering*, Vol. 9, No. 1, January 1983, pp. 57-68.

[BGWR81] Bernstein P. A., Goodman N., Wong E., Reeve C. L., Rothnie J. B. Query Processing in a System for Distributed Databases (SDD-1), *ACM Transactions on Database Systems*, Vol. 6, No. 4, December 1981, pp. 602-625.

[BreT85] Breitbart Y. J., Tieman L. R., ADDS-Heterogeneous Distributed Database System, in *Distributed Data Sharing Systems*, Schreiber F. A., Litwin W. (Eds), Elsevier Science Publishers, North-Holland, 1985.

[CerP85] Ceri S., Pelagatti G., *Distributed Databases: Principles & Systems*, McGraw-Hill, New York, 1985.

[ChuH82] Chu W. W., Hurley P., Optimal Query Processing for Distributed Database Systems, *IEEE Transactions on Computers*, Vol. 31, No. 9, September 1982, pp. 135-150.

[Corn88] Cornelis B., Site Autonomy in a Distributed Database Environment, IEEE Spring COMPCON 1988, *Thirty-Third IEEE Computer Society International Conference*, 1988, San Francisco, CA, USA, pp. 440-443.

[Daya85] Dayal U., Query Processing in a Multidatabase System, in *Query Processing in Database Systems*, Kim W., Reiner D. S., Batory D. S. (Eds.), Spring-Verlag, Berlin, 1985.

[DuKS92] Du W., Krishnamurthy R., Shan M.C. Query Optimization in Heterogeneous DBMS, *Proceedings of the 18th VLDB Conference* pp. 277-299, 1992

[GarK88] Garcia-Molina H., Kogan B., Node Autonomy in Distributed Systems, *Proceedings International Symposium on Databases in Parallel and Distributed Systems*, 1988, pp. 158-166, In: Multidatabase Systems: An Advanced Solution for Global Information Sharing, Hurson A. R., Bright M. W., Pakzad S. H. (Eds.), IEEE Press, Washington, 1994.

[GarJ79] Garey M.R., Johnson D.S. Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., San Francisco, 1979.

[HorS78] Horowitz E., Sahni S. Fundamentals of Computer Algorithms, Computer Science Press, Inc., 1978.

[HaSN96] Harangsri B., Shepherd J., Ngu A. Query Classification in Multidatabase Systems, Proceedings of the 7th Australasian Database Conference, ADC'96, pp.147- 156, 1996

[LimS93] Lim E-P., Srivastava J., Query Optimization and Processing in Federated Database Systems, *2nd International Conference on Information and Knowledge Management, CIKM*'93 Washington DC, 1993.

[LuOG93] Lu H., Ooi B-C., Goh C-H., Multidatabase Query Optimization: Issues and Solutions, *Proceedings RIDE-IMS'93, Research Issues in Data Engineering: Interoperability in Multidatabase Systems*, April 1993, Vienna, Austria, pp. 137-143.

[LuSh92] Lu H., Shan M-C., On Global Query Optimization in Multidatabase Systems, *Second International Workshop on Research Issues on Data Engineering: Transaction and Query Processing*, Tempe, Az, USA, February. 1992, pp. 217.

[RePR89] Reddy M. P., Prasad B. E., Reddy P. G., Query Processing in Heterogeneous Distributed Database Management Systems, in *Integration of Information Systems: Bridging Heterogeneous Databases*, Gupta A. (Ed.), IEEE Press, New York, 1989, pp. 264-278.

[SchC92] Scheuermann P., Chong E. I., Query Processing in Multidatabase Systems, *International Conference on Systems, Man and Cybernetics*, New York, 1992, pp. 1250-1254.

[SheL90] Sheth A. P., Larson J. A., Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases, *ACM Computing Surveys*, vol. 22, no. 3, September 1990, pp. 183-236.

[Ullm89] Ullman J.D. Principles of Database and Knowledge-Base Systems, vol. 2, pp. 662-668, Computer Science Press, 1989

[YuLi90] Yu C., Liu C., Experiences with Distributed Query Processing, *Sixth International Conference on Data Engineering*, Los Angeles, CA, USA, February 1990, pp. 192-199.

[ZhuL94] Zhu Q., Larson P.A., A Query Sampling Method for Estimating Local Cost Parameters in a Multidatabase System, Data Engineering, pp. 144-153, 1994.

# A Appendix

Reductions for intersection, union and difference operations are given in tables 5, 6, 7, 8.

| $\cap$ | $A(q_j)$ | $A(q_j)+$ | $N_y(q_j)+$ | $E_y(q_j)$ | $E_y(q_j)+$ |
|---|---|---|---|---|---|
| A($q_i$) | $\sigma_{\phi(q_i)}(q_j)$ | $\sigma_{\phi(q_i)}(q_j)$ | $\sigma_{\phi(q_i)}(q_j)$ | $\sigma_{\phi(q_i)}(q_j)$ | $\sigma_{\phi(q_i)}(q_j)$ |
| A($q_i$)+ | NIL | NIL | NIL | NIL | NIL |
| $N_x$($q_i$)+ | NIL | NIL | NIL | NIL | NIL |
| $E_x$($q_i$) | $\sigma_{\phi(q_i)}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ |
| $E_x$ ($q_i$)+ | NIL | NIL | NIL | NIL | NIL |

Table 5: Reductions for intersection operation (k = x $\cap$ y).

| $\cup$ | $A(q_j)$ | $A(q_j)+$ | $N_y(q_j)+$ | $E_y(q_j)$ | $E_y(q_j)+$ |
|---|---|---|---|---|---|
| A($q_i$) | $\sigma_{\phi}(q_i)(q_j)$ | $\sigma_{\phi}(q_i)(q_j)$ | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ |
| A($q_i$)+ | $\sigma_{\phi}(q_i)(q_j)$ | $\sigma_{\phi}(q_i)(q_j)$ | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ |
| $E_x$($q_i$) | NIL | NIL | NIL | NIL | NIL |
| $E_x$($q_i$)+ | NIL | NIL | NIL | NIL | NIL |

Table 6: Reductions for union operation (k = x $\cap$ y).

| $r-$ | $A(q_j)$ | $A(q_j)+$ | $N_y(q_j)+$ | $E_y(q_j)$ | $E_y(q_j)+$ |
|---|---|---|---|---|---|
| A($q_i$) | $\sigma_{\phi}(q_i)(q_j)$ | $\sigma_{\phi}(q_i)(q_j)$ | $\sigma_{\phi}(q_i)(q_j)$ | $\sigma_{\phi}(q_i)(q_j)$ | $\sigma_{\phi}(q_i)(q_j)$ |
| A($q_i$)+ | NIL | NIL | NIL | NIL | NIL |
| $N_x$($q_i$)+ | $\sigma_{\phi}(q_i)(q_j)$ | NIL | NIL | NIL | NIL |
| $E_x$($q_i$) | $\sigma_{\phi}(q_i)(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{\phi(\pi_k(q_i))}(q_j)$ |
| $E_x$($q_i$)+ | NIL | NIL | NIL | NIL | NIL |

Table 7: Reductions for difference operation when a reduced subquery is on the right hand side of difference operation (k = x $\cap$ y).

| $l-$ | $A(q_j)$ | $A(q_j)+$ | $N_y(q_j)+$ | $E_y(q_j)$ | $E_y(q_j)+$ |
|---|---|---|---|---|---|
| A($q_i$) | $\sigma_{not\phi(q_i)}(q_j)$ | $\sigma_{not\phi(q_i)}(q_j)$ | NIL | NIL | NIL |
| A($q_i$)+ | $\sigma_{not\phi(q_i)}(q_j)$ | $\sigma_{not\phi(q_i)}(q_j)$ | NIL | NIL | NIL |
| $N_x$($q_i$)+ | $\sigma_{not\phi(q_i)}(q_j)$ | NIL | NIL | NIL | NIL |
| $E_x$($q_i$) | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ | NIL | NIL | NIL |
| $E_x$($q_i$)+ | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ | $\sigma_{not\phi(\pi_k(q_i))}(q_j)$ | NIL | NIL | NIL |

Table 8: Reductions for difference operation when a reduced subquery is on the left hand side of difference operation (k = x $\cap$ y).