Processing Queries Over Generalization Hierarchies in a Multidatabase System

Umeshwar Dayal

Computer Corporation of America
Four Cambridge Center
Cambridge, Massachusetts 02142

Abstract

An important task of multidatabase systems is the integration of existing databases. Database integration is achieved primarily through the use of generalization. Hence, it is important to develop good tactics for processing queries over generalization hierarchies. This paper defines the class of conjunctive generalization queries, and it describes four tactics for processing these queries that have been developed for the MULTIBASE system. Since query processing tactics are best describe algebraically, the paper shows how to model generalization as a sequence of algebraic operations. Three of the tactics described here are adapted from conventional distributed query processing techniques. However, it is argued that these tactics are of limited applicability to processing queries over generalization hierarchies. A fourth tactic, semiouterjoin, which is more widely applicable, is introduced.

This research was jointly supported by the Defense Advanced Research Projects Agency of the Department of Defense and the Naval Electronic Systems Command and was monitored by the Naval Electronic Systems Command under Contract No. N00039-82-C-0226. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the DARPA, NAVELEX, or the U.S. Government.

1. Introduction

A multidatabase management system such as MULTIBASE [SBDG81, LR82] is a system that provides uniform, integrated access to a heterogeneous distributed collection of databases. It differs from a conventional distributed database management system (e.g., SDD-1 [RBFG80], System R* [WDHL82], Distributed INGRES [Ston77], DDM [CDFG83]) in two significant respects. First, the databases are heterogeneous, i.e., stored under different local database management systems, each with its own data model and language. Second, the databases are preexisting, i.e., have been designed and maintained independently of one another, and hence may be inconsistent. In [KG81, DH82] we describe how MULTIBASE shields users from these problems of heterogeneity and inconsistency. The local databases are first 'homogenized' by describing their schemas in a common data model, DAPLEX [Ship 81]. Database integration is then achieved by defining a global view tailored to the user's application over these DAPLEX representations; the view definition incorporates directives for resolving differences between the local databases.

A user formulates queries in DAPLEX over his global view. The processing of a global query consists of four tasks:

1. Query modification. The global DAPLEX query is modified into a DAPLEX query over the local schemas.

2. Global query optimization. A global execution plan is constructed for the modified query. The plan is composed of <u>single-site queries</u> (each posed against exactly one local schema), <u>move</u> operations that ship results of the single-site queries between sites, and <u>postprocessing queries</u> that integrate the results of the single-site queries. In MULTIBASE, query modification, global query optimization, and the execution of postprocessing queries are performed at a special <u>global site</u>.

3. Local query optimization. The single-site queries sent to each local site are sub-

jected to local access path optimization.

4. Translation. The optimized queries at each site are translated into equivalent queries or programs in the data language of the host DBMS.

Query modification has been described in [KG81, DH82], and local query optimization in [DGLO81, DG82]. Here, we focus on global optimization.

There are several novel aspects of global query optimization in a multidatabase system that are not present in conventional distributed query optimization [BGWRR81, HY79, SA80, YO79, ESW78]. These are direct consequences of heterogeneity and database integration. Heterogeneity is manifested in two ways: differences in the relative processing speeds of the sites, and differences in the capabilities of the sites. In MULTIBASE, the speeds and capabilities of the sites are provided as parameters to the global optimizer. When the global optimizer compares various alternative global execution plans, it factors the differences in speeds into its cost evaluation. Also, it ensures that all queries sent to a site can be processed there. Thus, if the DBMS at a site cannot compute joins, process quantifiers, etc., queries bound for that site must be either decomposed into subqueries that can be processed there, or 'filtered' to remove parts of the query that can be processed (in which case appropriate compensatory postprocessing queries must be added to the plan).

Database integration has a more profound impact on global query optimization. The principal technique that we propose for the integration of databases that contain data about similar objects is <u>generalization</u> [SS77, KG81, DH82]. For example, consider two Ship databases. Suppose that different attributes are defined for the Ship entities in the two databases. (To distinguish between the Ship entity types in the two different local schemas, LS1 and LS2, we refer to them as LS1.Ship and LS2.Ship.) In the global schema, we can define a generalization hierarchy consisting of LS1.Ship, LS2.Ship, and a generic entity type, Ship. The attributes of Ship are the common attributes of the two subtypes. We now have to efficiently process queries over this generalization hierarchy. We are not aware of any previous solutions to this problem. The main contribution of this paper is that it develops such a solution.

If the databases are disjoint, the solution is straightforward. Thus, in our example, the logical Ship file in the view may be thought of as the union of the Ship files (appropriately projected) in the two local databases. Strategies for processing queries against horizontally partitioned files have been developed before [BGWRR81, SA80, ESW78]. In [DLY82] we show how to adapt and extend these strategies for our purpose.

However, the more interesting (and difficult) problems occur when the databases overlap. The definition of the global view must then specify a <u>merge condition</u> under which entities in different databases are to be considered as the same logical entity. For example, (s1 in LS1.Ship == s2 in LS2.Ship when HullNo (s1) = IdNo (s2)) is a merge condition specifying that an LS1.Ship entity is 'the same as' an LS2.Ship entity iff the HullNo of the former is equal to the IdNo of the latter. Now the logical Ship file in the view is no longer the union of the local Ship files but their 'outerjoin'.

Further complications arise when the databases are inconsistent, i.e., disagree on the attribute values of some entity. For example, the first database might record the value 50 for the deadweight of the ship that has HullNo 1234, while the second database records the value 60 for the deadweight of the ship that has IdNo 1234; but in the view these local entities represent the same logical Ship entity. The discrepancy is resolved in the view definition by an appropriate <u>aggregate function</u>; for instance, the deadweight in the view might be specified to be the average of the deadweights in LS1 and LS2. So the problem of processing queries over generalization hierarchies in a multidatabase system is related to the problem of processing aggregates.

In this paper we develop tactics for processing selection, projection, and join queries over generalization hierarchies in which some attributes of a generic entity type are defined by aggregation from attributes of its subtypes. In Section 2, we define this class of queries, which we call <u>conjunctive generalization queries</u>.

Query processing tactics typically are based on algebraic properties. Hence, in Section 3, we define generalization algebraically in terms of outerjoins and aggregates, and construct algebraic equivalents of conjunctive generalization queries.

In Section 4, four tactics for processing these queries are developed. Two of these tactics are for efficiently processing selections. The reader might find this surprising, because processing selections (and projections) in a conventional distributed database system is easy: they can be processed completely locally at a single site. Even if some files are horizontally partitioned, it is easy to process a selection and projection query: execute the same query locally at each of the sites containing a horizontal fragment, and then construct the union of the partial results. (This is why most previous research on distributed query optimization ignored selections and projections, and focused on joins [BGWRR81, SA80, HY79, YO79, ESW78].)

When aggregates occur in view definitions, this simple approach no longer works. Consider a query that selects all ships whose deadweight exceeds 55. In our example, the selection clearly cannot be done locally at each site because it is necessary to compute the average of the two deadweights. However, if instead of the average, the deadweight in the view was defined to be the maximum of the local deadweights, then local selection is possible. Since 'localizing' selections reduces the volume of the data moved, we develop two tactics for it. One is to analyze special aggregate functions and to determine for them whether the selection can be distributed over the generalization. The other tactic (which works for all aggregates) is to perform 'semiouterjoin reductions' (called 'semiunion reductions' in [DR82]). In our example, the reduction of LS1.Ship by LS2.Ship requires shipping IdNo values from the latter site to the former. The LS1.Ship entities are then partitioned into two subsets: those contained only in the first database, (i.e., the subset $\{s1 \in LS1.Ship | (\forall s2 \in LS2.Ship) (HullNo(s1) \neq IdNo(s2))\}$), and those having corresponding entities in the second database. For the first subset, the selection can be performed locally; the second subset must be retrieved to perform the aggregate and selection by postprocessing at the global site. The other two tactics developed in Section 4 are for processing joins.

In Section 5, we describe briefly how to use these four tactics to construct global execution plans. For a cost model and further details of global query optimization in MULTIBASE, the reader is referred to [DLY82].

To summarize, the main contribution of this paper is that it extends traditional relational query processing techniques, which focus on select-project-join queries, to techniques for processing queries containing outerjoins and aggregates.

2. Conjunctive Generalization Queries

We first define conjunctive DAPLEX queries, which correspond roughly to the conjunctive relational queries considered by [BGWRR81, HY79, SA80, YO70, ESW78]. We then extend this class to conjunctive generalization queries.

2.1 Conjunctive DAPLEX Queries and Query Graphs

A database schema in the DAPLEX Model is a directed multigraph, whose nodes are entity types, and whose edges are single-valued or multi-valued functions. For our purpose, a conjunctive DAPLEX query is in the following canonical form:

```
for each $x_1$ in $X_1$
 for each $x_2$ in $X_2$
 .
  .
   .
    for each $x_n$ in $X_n$
        where <qualification>
        output <target_list>
    endfor
   .
  .
 .
 endfor
endfor
```

The X_i are entity types. The target_list is a list of terms f(x), where x is one of the iteration variables $x_1, x_2, \ldots, x_n$, and f is a single-valued function whose range is a set of scalars (e.g., Real, Integer, String, Boolean). The qualification is a conjunction of atomic formulas of the following types: one-variable selection clauses: $(f(z) \text{ op } c)$, $(f_1(z) \text{ op } f_2(z))$, $(c \text{ isin } h(z))$; value-based join clauses: $(f(z) \text{ op } g(w))$; and linked join clauses: $(w = f(z))$, $(w \text{ isin } h(z))$; where w, z are variables; c is a constant; f, f_1, f_2, g are single-valued functions; h is a multivalued function; and op is one of the arithmetic comparison operations (e.g., $=$, $\leq$, $\neq$).

The query graph of a conjunctive query is an undirected graph that has one node for each variable occurring in the query, and one edge $\{w,z\}$ for each set of join clauses involving variables w and z in the qualification. Attached to each node are any one-variable selection clauses (labeled σ) and target-list functions (labeled π) defined on the corresponding variable. Each edge is labeled with the corresponding join clauses.

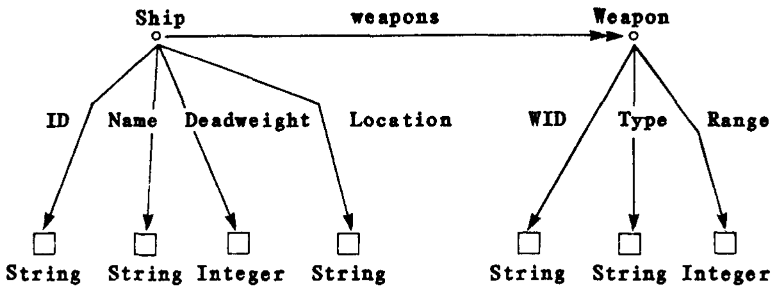
Examples of a DAPLEX schema, a conjunctive query and its query graph are given in Figure 2.1.

2.2 Conjunctive Generalization Queries

A conjunctive generalization query is a conjunctive query, some of whose variables may range over generalization hierarchies. A conjunctive generalization query is produced by modifying a conjunctive query containing a variable that ranges over a generic entity type in a view, where the generic entity type's functions are defined in terms of its subtypes' functions. The details of view definition and query modification in DAPLEX are irrelevant to our discussion here (see [KG 81, DH 82, DLY 82]).

The query graph of a conjunctive generalization query is very similar to that of a conjunctive query, except that variables ranging over generalization hierarchies are represented by

Schema:



Query:

```
for each s1 in Ship
  for each s2 in Ship
    for each w1 in Weapon
      for each w2 in Weapon
        where w1 isin weapons(s1) AND Deadweight(s1)>55 AND
              Location(s1) = Location(s2) AND
              w2 isin weapons(s2) AND
              Type(w1) = Type(w2) AND Range(w2)>Range(w1)
        output ID(s1), Name(s1), WID(w1)
      endfor
    endfor
  endfor
endfor
```

Query Graph:

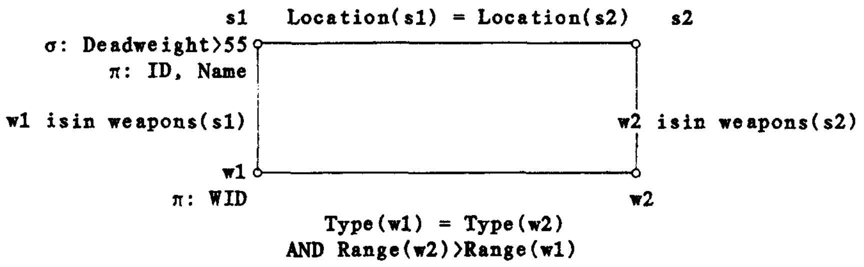


Figure 2.1 A Conjunctive DAPLEX Query and its Query Graph

generalization nodes. A generalization node is labeled with the corresponding generalization hierarchy, a merge condition on the subtypes, and a SubRange Table(SRT) that encapsulates the definition of the functions on the generic entity type in terms of the functions on its subtypes. The SRT in Figure 2.2, for example, shows that (a) for each entity in LS1.Ship that has no corresponding entity in LS2.Ship, the Deadweight in the view is equal to its Deadweight1 value in the LS1 database; (b) symmetrically, for each entity in LS2.Ship that has no matching LS1.Ship entity, the Deadweight is equal to its Deadweight2 value in LS2; and (c) for each ship that is represented in both LS1.Ship and LS2.Ship, the Deadweight is the maximum of its two deadweight values in LS1 and LS2.

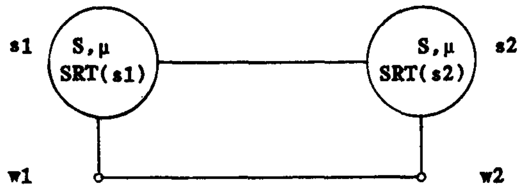
3. Algebraic Equivalents of Conjunctive Generalization Queries

3.1 Modeling a DAPLEX Schema

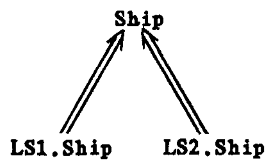
We model a DAPLEX schema by a relational schema as follows. For each entity type there is one relation whose attributes are exactly the functions defined on that entity type. (We shall

Suppose that in the example of Figure 2.1, Ship is a generic entity type defined over subtypes LS1.Ship and LS2.Ship. Assume that for each function f on Ship, the corresponding functions in LS1.Ship and LS2.Ship are named f1 and f2, respectively. Then the query of Figure 2.1 becomes a conjunctive generalization query after modification.

Query graph:



Node and edge labels are identical to those in Figure 2.1. S is the generalization hierarchy



μ is the merge condition: (x1 in LS1.Ship == x2 in LS2.Ship when HullNo(x1) = IdNo(x2))

SRT(si) for i = 1,2:

	Id	Deadweight	Name
LS1.Ship - LS2.Ship	HullNo	Deadweight1	Name1
LS2.Ship - LS1.Ship	IdNo	Deadweight2	Name2
LS1.Ship ∩ LS2.Ship	HullNo|IdNo	max(Deadweight1, Deadweight2)	Name1|Name2

	Location	Weapons
LS1.Ship - LS2.Ship	Location1	weapons1
LS2.Ship - LS1.Ship	Location2	weapons2
LS1.Ship ∩ LS2.Ship	avg(Location1,Location2)	weapons1∪weapons2

Note:

1. LS1.Ship - LS2.Ship denotes $\{s \in \text{Ship} \mid (\exists s1 \in \text{LS1.Ship})(\text{ID}(s) = \text{HullNo}(s1))$ $(\forall s2 \in \text{LS2.Ship})\ (\text{HullNo}(s1) \neq \text{IDNo}(s2))\}$
2. LS1.Ship ∩ LS2.Ship denotes $\{s \in \text{Ship} \mid (\exists s1 \in \text{LS1.Ship})(\exists s2 \in \text{LS2.Ship})$ $(\text{ID}(s) = \text{HullNo}(s1) = \text{IDNo}(s2))\}$
3. Name1|Name2 denotes 'either Name1 or Name2'

Figure 2.2 Query Graph of a Conjunctive Generalization Query

use X.A to denote attribute A of relation X.) Assume that each entity type X has a unique identifier function X.ID. The relations are populated by one tuple per entity: if f is a function from X to a scalar range, then for entity $x \in X$, the corresponding tuple x has X[X.f] = f(x); if f is a function from X to entity set Y, then for entity $x \in X$, the corresponding tuple x has x[X.f] = ID(f(x)) when f is single-valued, and x[X.f] = $\{\text{ID}(y) \mid y \in f(x)\}$ when f is multivalued.

Note that because of multivalued functions, the resulting relations are not in First Normal Form (1NF).

3.2 Modeling a Conjunctive DAPLEX Query

A conjunctive DAPLEX query can be modeled algebraically using the relational operations of Cartesian product (or join), selection, and projection.* The mapping is essentially the same as for Relational Calculus [Codd72], QUEL [HSW75], or SQL [CAEG76]. Thus, the canonical conjunctive DAPLEX query of Section 2.1 is equivalent to the algebraic expression $\pi_T E_Q$, where π_T denotes projection on the target list and E_Q is an algebraic expression constructed from the qualification Q by first forming the Cartesian product P of all entity sets referenced by the query, and then restricting P by selections corresponding to the clauses in Q. (Note that the operations of relational algebra apply equally well to non-1NF relations [JS82]. For non-1NF relations, the following selection and join conditions are applicable in addition to those listed in [Codd70]: $\sigma_{a \in A} R$, which corresponds to the selection clause a isin A(r); and S[S.ID $\in$ R.f]R, which corresponds to the linked join clause s isin f(r).) Applying this procedure to the query in Figure 2.1 yields:

$$\pi_{s1.ID, s1.Name, w1.WID}\ \sigma_{w1.ID \in s1.weapons}$$

$$\sigma_{s1.Deadweight>55}\ \sigma_{s1.Location = s2.Location}$$

$$\sigma_{w2.ID \in s2.Weapons}\ \sigma_{w1.Type = w2.Type}$$

$$\sigma_{w2.Range>w1.Range}\ (s1 \times w1 \times s2 \times w2)$$

where s1, s2 are copies of the Ship relation, and w1, w2 are copies of the Weapon relation.

3.3 Modeling Generalization

We define generalization algebraically in two steps. First, construct the outerjoin of the subtype relations on the merge condition. (Informally, the outerjoin of R and S on condition μ, denoted $R(\mu)S$, is the union of the join $R[\mu]S$ together with tuples constructed from the unjoined tuples of R and the unjoined tuples of S by padding them out with NULL values [Codd79].)

The second step is to define the attributes of the generic relation by aggregation over the attributes of its subtypes. Informally, we proceed as follows. Consider the outerjoin of LS1.Ship and LS2.Ship. Add a column (an attribute) Deadweight to this relation; the value of Deadweight in each tuple is the maximum of the Deadweight1 and Deadweight2 values of the tuple. Delete the Deadweight1 and Deadweight2 columns. Once this is done for all generic attributes, the resultant relation is Ship. Proceeding formally, we introduce an aggregate operation as follows. Let R be a relation over attributes $\underline{R}$. Let $X = \{A_1, \ldots, A_n\} \subseteq \underline{R}$. Denote domain $(A_1) \times \ldots \times$ domain(A_n) by domain(X). Let A be an attribute name not in $\underline{R}$, and $agg_A[X]$ be a function: domain(X) --> domain(A). Then, we extend $agg_A[X]$ to apply to the relation R in the obvious way; i.e., $agg_A[X]$ is a relation over $(\underline{R} - X) \cup \{A\}$ defined by: for each tuple $r \in R$, there is a tuple t in $agg_A[X](R)$, such that $t[\underline{R} - X] = r[\underline{R} - X]$ and $t[A] = agg_A(r[X])$. We call $agg_A[X]$ an aggregate function over $\underline{R}$. (Often we shall drop X from the name of the function, if it is clear from the context.) Some aggregate functions that will be of interest to us are listed in Figure 3.1.

Thus, generalization can be expressed as a sequence of outerjoin and aggregate operations. (If the subtypes being generalized are disjoint, then the outerjoin reduces to the outerunion [Codd79], and aggregation is unnecessary.) For the example of Figure 2.2, we can write:

$$\text{Ship} := \text{chooseany}_{Id}\ [\text{HullNo, IdNo}]$$

$$\text{chooseany}_{Name}\ \max_{deadweight}$$

$$\text{average}_{Location}\ \text{chooseall}_{Weapons}$$

$$(\text{LS1.Ship (HullNo = IdNo) LS2.Ship})$$

3.4 Modeling Conjunctive Generalization Queries

A conjunctive generalization query is similar to the canonical conjunctive DAPLEX query of Section 2.1, except that some of the X_i's may be the result of generalization. In the algebraic expression corresponding to the query, replace X_i by the outerjoin-aggregation expression defining X_i.

Applying this procedure to the query in Figure 2.2 yields the same expression as in Section 3.2, except that now s1 and s2 are copies of Ship as defined in Section 3.3.

4. Tactics for Processing Conjunctive Generalization Queries

In Section 3 we showed how to construct an algebraic expression equivalent to a conjunctive (generalization) query. A straightforward approach to processing a query would be to retrieve all the relations appearing in this expression to a result site (the global site in MULTIBASE) and then to directly evaluate the expression there. However, this direct approach would be extremely inefficient in general.

In practice, algebraic identities are used to transform the expression into equivalent

*This is only partly true, since DAPLEX queries do not usually remove duplicates. (SQL [CAEG76] has similar semantics.) To be accurate, therefore, we must use a multiset relational algebra to model DAPLEX [DGK82]. However, for simplicity, we describe our query processing tactics in terms of relational algebra. They apply equally well to multiset relational algebra.

1. max, min, sum, count, average: Let R be a relation over $\underline{R}$, and A1, A2 $\in \underline{R}$. Then $\max_A$[A1,A2] is the aggregate function defined by:

$$\max_A [A1,A2](r) = \begin{cases} r[A1], \text{ if } r[A1] \neq \text{null}, r[A2] = \text{null} \\ r[A2], \text{ if } r[A1] = \text{null}, r[A2] \neq \text{null} \\ \max(r[A1], r[A2]), \text{ otherwise} \end{cases}$$

The other aggregate functions are defined similarly.

2. choose1: Let R, $\underline{R}$, A1, A2 be as before. Then,

$$\text{choose1 } [A1;A2](r) = \begin{cases} r[A1], \text{ if } r[A1] \neq \text{null} \\ r[A2], \text{ if } r[A1] = \text{null} \end{cases}$$

choose2 is defined symmetrically.

3. chooseany: Let R, $\underline{R}$, A1, A2 be as before. Then,

$$\text{chooseany}_A [A1,A2](r) = \begin{cases} r[A1], \text{ if } r[A1] \neq \text{null}, r[A2] = \text{null} \\ r[A2], \text{ if } r[A1] = \text{null}, r[A2] \neq \text{null} \\ r[A1] \text{ (or } r[A2]), \text{ if } r[A1] = r[A2] \\ \text{null otherwise} \end{cases}$$

4. chooseall: Let R, $\underline{R}$, A1, A2 be as before. Then,

$$\text{chooseall}_A [A1,A2](r) = \begin{cases} r[A1], \text{ if } r[A1] \neq \text{null}, r[A2] = \text{null} \\ r[A2], \text{ if } r[A1] = \text{null}, r[A2] \neq \text{null} \\ \{r[A1], r[A2]\} \text{ otherwise} \end{cases}$$

If A1, A2 are multivalued then $\text{chooseall}_A [A1,A2](r) = r[A1] \cup r[A2]$.

These definitions can be easily extended to more than two attributes A1, A2, A3 ....

Figure 3.1 Some Common Aggregate Functions

expressions that potentially are cheaper to process. First, the product-selection expression E_Q is replaced by an equivalent expression involving joins and selections as follows. Consider any spanning tree of the query graph. For each edge e of the spanning tree, replace the product R1 X R2 and the selection σ_J by the join R1[J]R2, where R1,R2 are the endpoints of e and J is the join condition labeling e.

Other transformations are based on various tactics for reducing the volume of data moved between sites. (Our cost function is a weighted sum of data movement costs and local processing costs.) The tactics usually entail a tradeoff between reduced data movement and increased local processing. Two commonly used tactics for processing conjunctive queries in conventional distributed systems are: (1) performing selections (and projections) locally at individual sites before performing any inter-site join; and (2) semijoin reductions [BGWRR81]. These tactics are applicable even when some of the relations are horizontally partitioned. The usual approach is to transform the query into the union of a collection of conjunctive subqueries. For example, if R1 is partitioned into R1,R2, and S is partitioned into S1,S2, then the query

$$\pi_T \sigma_{R.A = a} \sigma_{S.B = b} (R[J]S)$$

is transformed into

$$\bigcup_{i,j \in \{1,2\}} \pi_T \sigma_{Ri.A = a} \sigma_{Sj.B = b} (Ri[J]Sj).$$

Each subquery is then separately optimized using the two tactics listed above. This approach works because selection, projection, and join distribute over union.

However, these tactics are not always applicable to conjunctive generalization queries that involve selection, projection, or join over aggregated attributes. Below, we investigate the distribution of selection and join over generalization. Then we describe conditions under which semijoin reductions can be used. Finally, observing that these three 'conventional' tactics can be used only in rather special cases, we introduce a fourth tactic, semiouterjoin reduction, which plays an important role in processing generalization queries.

Tactic 1: Distributing Selection over Generalization

Let R1,R2 be subtypes of R. Consider the selection query $\sigma_{A\ op\ a}R$. We remarked in Section 3.3 that generalization over disjoint subtypes can be modelled as the outerunion operation. In this case, selection can, indeed, be distributed over the generalization, since $\sigma(R1(+)R2) = \sigma R1(+)\sigma R2$, where (+) denotes outerunion. This means that we can perform the selection locally at R1's site and at R2's site, and then merge the results.

However, when the subtypes overlap and the selection clause is on an aggregated attribute, the following equality holds sometimes, but does not hold in general: $\sigma_{A\ op\ a}\ agg_A\ (R1(\mu)R2) = agg_A\ (\sigma_{A1\ op\ a}\ R1(\mu)\sigma_{A2\ op\ a}\ R2)$ where μ is the merge condition on R1 and R2. Whenever equality holds, it is possible to perform selection locally, and then generalize the results.

For the example in Section 1, equality holds for the max aggregate function and the given query, which selects on deadweight > 55. However it does not hold if we replace the max aggregate function in the definition of deadweight by the avg aggregate function, or if we change the selection clause in the query to deadweight = 55.

In some cases, through strict distributivity does not hold, it is possible to perform a modified selection at each site, and then perform a postselection after generalization (in practice, the postselection and aggregation can be implemented together in one pass through the outerjoin file); i.e., it is possible to find op1, op2 such that:

$$\sigma_{A\ op\ a}\ agg_A(R1(\mu)R2) = \sigma_{A\ op\ a}\ agg_A(\sigma_{A1\ op1\ a}\ R1(\mu)\ \sigma_{A2\ op2\ a}\ R2).$$

For example, if the max aggregate is used to define deadweight and the selection query is deadweight = 55, then we can perform local selections on the modified condition deadweight $\geq$ 55, and the postselection on deadweight = 55. For the average aggregate function, even this approach is not applicable. Figure 4.1 lists some cases in which this tactic can be used.

The benefit of this tactic is the reduction in the volume of data moved: $\sigma_{Ai\ opi\ a}\ Ri$ generally is smaller than Ri.

Tactic 2: Distributing Joins over Generalization

In conventional distributed systems that support horizontal partitioning, joins are always distributed over unions [BGWRR81, SA80]. Thus, if R1, R2 are horizontal fragments of R, and if S1, S2 are horizontal fragments of S, then the query R[J]S is always replaced by the union of four subqueries Ri[J]Sj, $1 \leq i,\ j \leq 2$. Actually, in some instances this might not be the cheapest strategy. It is sometimes cheaper to first construct R or S (or both) using unions, and then perform the join. We treat distribution of joins as a tactic to be used only if it is beneficial. In a multidatabases system, where R1, R2 are subtypes of R, and S1, S2 are subtypes of S, distributing joins over generaliztion may not even be possible. Hence, we first have to consider whether the tactic is applicable before we assess its benefit.

Let R1,R2 be subtypes of R, and let S1,S2 be subtypes of S. Consider the join R[A op B]S. When the subtypes of R are disjoint, the join can be left-distributed over the outerunion, thus: R1[A1 op B]S (+) R2[A2 op B]S. This means that it is possible to join S separately with R1 and R2, and then merge the results. (Symmetric, for right-distribution.)

However, when the subtypes overlap and the join condition is on aggregated attributes, one or the other (or both) types of distributivity may not hold. Given the results of Figure 4.1, it is not surprising that left distributivity holds when agg_A is $chooseany_A$ or $chooseall_A$ (symmetric for right distributivity). For other aggregate functions, we could analyze the combination of agg_A, agg_B, and op, to find a modified op for each join Ri[Ai op Bj]Sj, and to determine the postprocessing required to produce the correct result. To keep the query processing strategies simple, however, we decided not to do this. Instead of distributing joins over generalization, we distribute only semijoins [BC81].

The semijoin R<A op B]S is the set of R-tuples that will join with at least one S-tuple, and is equivalent to a union of selections

$$\bigcup_{b \in \pi_B S} \sigma_{A\ op\ b} R.$$

Thus, to determine if the semijoin can be left-distributed, we can use the tables in Figure 4.1 (for each b).

Distribution is advantageous in two cases. The first occurs when the subtypes of both R and S are disjoint, site(Ri) = site(Si) for all i, and the 'cross-terms' Ri[Ai op Bj]Sj = $\emptyset$ for all $i \neq j$. Now, the join of R and S reduces to the completely local joins of Ri and Si (for all i), followed by the outerunion of their results. The second case occurs when the semijoin reduction tactic is beneficial. We discuss this tactic next.

Tactic 3: Semijoin Reduction

The semijoin R<A op B]S is executed by retrieving $\pi_B S$ at S's site, shipping it to R's site, and then restricting R. This tactic was

$$\sigma_{A\ op\ a}\ agg_A\ (R1(\mu)R2) = \sigma_{A\ op\ a}\ agg_A(\sigma_{A1\ op1\ a}R1(\mu)\ \sigma_{A2\ op2\ a}R2)$$

Case 1. agg = chooseany [A1,A2] or chooseall [A1,A2]: True distributivity holds

op1 = op2 = op

Case 2a. agg = max[A1,A2]:

op	$>$	$=$	$<$
op1,op2	$>$	$\geq$	True

Case 2b. agg = min[A1,A2]:

op	$<$	$=$	$>$
op1,op2	$<$	$\leq$	True

Case 3. agg = count[A1,A2], sum[A1,A2], or average[A1,A2]:

for any op, op1 = op2 = True

Case 4. agg = choose1 [A1,A2]:

for any op, op1 = True
op2 = op

Note: op = True indicates no reduction is possible.
op1 = op2 = op implies that postselection is unnecessary.

Figure 4.1 Distribution of Selection Over Generalization

proposed in [BC81], and it is used by both SDD-1 and System R*. The utility of the semijoin as a tactic for distributed query processing is based on the following properties:

1. $R\langle A\ op\ B]S \subseteq R$

2. $(R\langle A\ op\ B]S)\ [A\ op\ B]S = R[A\ op\ B]S$

Property 1 says that the semijoin can reduce the size of R before R is shipped to the global site for final processing. Property 2 says that after the reduction, R retains all the tuples that could possibly participate in the join. Hence the semijoin reduction will be beneficial if the total cost of retrieving and shipping the joining field of S, and performing the semijoin at R's site, is offset by the reduction in the size of R.

When R and S are the results of generalization, the applicability of this tactic is limited. If the join is not distributable over the generalization, both R and S have to be materialized before they are joined. However, in MULTIBASE, this materialization occurs at the global site. But then, since R and S are already at the result site, the semijoin reduction of R by S (or vice versa) is useless. Hence, the semijoin tactic should be considered only when Tactic 2 is applicable.

Tactic 4: Semiouterjoin Reduction

We have seen that the first three tactics are seldom applicable when overlapping subtypes are generalized. Thus, there is the need for a tactic that can be widely used and that has the potential to reduce query processing cost.

Consider a selection query $\sigma_{A\ op\ a}$ R, where R = agg_A (R1(μ)R2). If we can 'demarcate the boundary' between the 'private' and 'overlap' parts of R1 (resp. R2), then we can perform the selection over the private part of R1 (resp. R2); only the overlap part will then have to be retrieved for aggregation. The overlap part of R1 is the set of R1-tuples that have matching R2-tuples, i.e., the semijoin of R1 by R2 on the merge condition μ. Then the private part is R1 - R1$\langle\mu]$R2. We call this the antijoin of R1 by R2 on μ. The semiouterjoin of R1 by R2 on μ is an operation that partitions R1 into the semijoin and the antijoin, i.e., R1$\langle(\mu)$S = {R1$\langle\mu]$R2, R1 - R1$\langle\mu]$R2}.

The utility of the semiouterjoin is based on the following identity: agg_A(R1(μ)R2) = (R1-R1$\langle\mu]$R2) (+) agg_A((R1$\langle\mu]$R2)(μ)R2). (Similarly, we can reduce R2 by R1, or both R1 and R2 by each other.) The selection $\sigma_{A\ op\ a}$ can now be performed against the private part. For the overlap part, we must use the tables of Figure 4.1 to determine if a modified selection can be performed before the generalization. For example, consider the query $\sigma_{Deadweight = 55}$Ship, where Deadweight is defined to be the avg aggregate. We can reduce LS1.Ship, then perform the selection Deadweight = 55 on the private part of LS1.Ship. Figure 4.1 shows that no modified selection is possible for the overlap part, which must be retrieved in toto. Let these results be T11 and T12. At the global site, T12 is generalized with the result of the query executed at LS2, and postselection is performed on the result of generalization. Finally, T11 is merged with the result of postselection.

The semiouterjoin can be implemented in much the same way as the semijoin, since the semijoin and antijoin can be computed simultaneously. Observe that by itself the semiouterjoin is useless. Its utility as a tactic is that it can reduce the cost of a subsequent selection (or join). Thus, a semiouterjoin R1⟨μ⟩R2 is beneficial if the cost of performing it is offset by the reduction in the size of R1 produced by a selection (or semijoin) that can now be performed locally on the private part of R1.

For example, suppose the sizes of LS1.Ship and LS2.Ship are 100 units each, and the size of the IdNo projection of LS2.Ship is 10 units. Also, suppose that 80% of the LS1.Ship entities are in its private part (and, hence, only 20% in the overlap part), and that the selectivity of the selection clause Deadweight = 55 is 10%. Let us assume that the cost of moving data is proportional to the volume of data moved, and that local processing costs are negligible compared to data movement costs. (We make these assumptions here only for simplicity. Our actual cost model is quite general [DLY82].) Then, if the semiouterjoin reduction tactic is not used, all of LS1.Ship will have to be moved to the global site at a cost of 100 units. If the tactic is used, π_{IdNo}LS2.Ship has to be moved from LS2's site to LS1's site at a cost of 10 units; then, the overlap part of LS1.Ship (20 units) and the private part of LS1.Ship restricted on the selection clause (0.10 x 80 = 8 units) have to be moved to the global site. Thus, the reduction tactic has an immediate benefit of 100 - 38 = 62 units. Additional benefit can be obtained by using the overlap part of LS1.Ship to reduce LS2.Ship. If we make similar assumptions about sizes and selectivities, then the additional benefit is given by: Cost of moving LS2.Ship to the global site - (Cost of moving the HullNo projection of LS1.Ship's overlap part to LS2's site + Cost of moving the overlap part of LS2.Ship to the global site + Cost of moving the restricted private part of LS2.Ship to the global site) = 100 - (2 + 20 + 0.10 x 80) = 70 units.

5. Conclusion

This paper studied the problem of global query optimization in MULTIBASE, a multidatabase management system. The main reason that this problem is more difficult than the distributed query optimization problems studied earlier arises from the need to integrate existing databases. Database integration is accomplished primarily through generalization; hence, it is important to develop good techniques for processing queries over generalization hierarchies.

We defined the class of conjunctive generalization queries by extending the class of conjunctive queries, which formed the basis of most previous research on query optimization. We adapted three commonly-used tactics for distributed query processing, and argued that their applicability to our problem is rather limited. We introduced a fourth tactic, semiouterjoin reduction, which can be more generally used. (For a theoretical treatment of semiouterjoin reduction, see [Hwang82].)

In [DLY82], we show how to construct global execution plans using these tactics, how to estimate the costs of alternative plans, and how to optimize (i.e., choose an inexpensive plan). Briefly, for each generalized entity type, we apply Tactic 1 whenever possible. Then we use dynamic programming to enumerate join orders. For each join in a join order, we consider using Tactics 2, 3, and 4 if they are applicable and immediately beneficial. In [DLY82], we also enhance this technique to process a wider variety of queries than the conjunctive generalization queries considered here.

Acknowledgement

The author wishes to thank Nat Goodman, Terry Landers, Dan Ries, and Laura Yedwab for their invaluable comments. The techniques presented in this paper are being implemented in a prototype of MULTIBASE.

6. References

[BC81]
Bernstein, P.A., and D.M. Chiu, 'Using Semi-joins to Solve Relational Queries,' JACM, Vol. 28, No. 1, January 1981, pp. 25-40.

[BG79]
Bernstein, P.A., and N. Goodman, 'Inequality Semijoins,' Technical Report CCA-79-28, Computer Corporation of America, Cambridge, Mass., December 1979.

[BGWRR81]
Bernstein, P.A., N. Goodman, E. Wong, C. Reeve, and J.B. Rothnie, 'Query Processing in a System for Distributed Databases (SDD-1),' ACM Trans. on Database Systems, Vol. 6, No. 4, December 1981, pp. 602-625.

[CAEG76]
Chamberlin, D.D., M.M. Astrahan, K.P. Eswaran, P.P. Griffiths, R.A. Lorie, J.W. Mehl, P. Reisner, and B.W. Wade, 'SEQUEL 2: A Uniform Approach to Data Definition, Manipulation, and Control,' IBM J. Res. and Dev. Vol. 20, No. 6, November 1976, pp. 560-575.

[CDFR83]
Chan, A., U. Dayal, S.A. Fox, N. Goodman, D. Ries, and D. Skeen, 'Overview of an Ada-Compatible Distributed Database Manager (DDM),' Proc. ACM-SIGMOD Conference, June 1983.

[Codd70]
Codd, E.F., 'A Relational Model of Data for Large Shared Data Banks,' CACM, Vol. 13, No. 6, June 1970, pp. 377-387.

[Codd72]
Codd, E.F., 'Relational Completeness of Database Sublanguages' in Database Systems, Courant Computer Science Symp. 6 (R. Rustin, ed.), Prentice-Hall, Englewood Cliffs, N.J., 1972.

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

[DGLO81]
Dayal, U., N. Goodman, T.A. Landers, K. Olson, J.M. Smith, and L. Yedwab, 'Local Query Optimization in MULTIBASE -- A System for Heterogeneous Distributed Databases,' Technical Report CCA-81-11, Computer Corporation of America, Cambridge, Mass., September 1981.

[DLY82]
Dayal, U., T.A. Landers, and L. Yedwab, 'Global Query Optimization in MULTIBASE: A System for Heterogeneous Distributed Databases,' Technical Report CCA-82-05, Computer Corporation of America, Cambridge, Mass., 1982.

[DG82]
Dayal, U., and N. Goodman, 'Query Optimization for CODASYL Database Systems,' Proc. ACM-SIGMOD Conference, June 1982, pp. 138-150.

[DGK82]
Dayal, U., N. Goodman, and R. Katz, 'An Extended Relational Algebra with Control over Duplicate Elimination,' Proc. ACM Symposium on Principles of Database Systems, March 1982, pp. 117-123.

[DH82]
Dayal, U., and H.Y. Hwang, 'View Definition and Generalization for Database Integration in MULTIBASE: A System for Heterogeneous Distributed Databases,' Proc. Sixth Berkeley Workshop on Distributed Database Management and Computer Networks, February 1982, pp. 203-238; to appear in IEEE Trans. on Software Engineering.

[DR82]
Dayal, U., and D. Ries, 'Research on Query Optimization at Computer Corporation of America,' Database Engineering, Vol. 5, No. 3, September 1982, pp. 33-37.

[ESW78]
Epstein, R., M. Stonebraker, and E. Wong, 'Distributed Query Processing in a Relational Database System,' Proc. ACM-SIGMOD Conference, May 1978.

[GD81]
Gouda, M., and U. Dayal, 'Optimal Semijoin Schedules for Query Processing in Local Distributed Database Systems,' Proc. ACM-SIGMOD Conference, April 1981, pp. 164-175.

[HY79]
Hevner, A.R., and S.B. Yao, 'Query Processing in Distributed Database Systems,' IEEE Trans. on Software Engineering, Vol. SE-5, No. 3, May 1979, pp. 177-187.

[HSW75]
Held, G.D., M.R. Stonebraker, and E. Wong, 'INGRES: A Relational Database System,' Proc. AFIPS NCC 1975, pp. 409-416.

[Hwang82]
Hwang, H.Y., 'Database Integration and Query Optimization in Multi-database Systems,' Ph.D. Diss., Dept. of Computer Sciences, University of Texas at Austin, Austin, Texas (in preparation).

[KG81]
Katz, R., and N. Goodman, 'View Processing in MULTIBASE -- A Heterogeneous Database System,' in Entity-Relationship Approach to Information Modeling and Analysis, (P.P. Chen, ed.), ER Institute, Saugus, Calif., 1981.

[JS82]
Jaeschke, G. and H.-J. Schek, 'Remarks on the Algebra of Non First Normal Form Relations,' Proc. ACM Symposium on Principles of Database Systems, March 1982, pp. 124-138.

[LR82]
Landers, T.A., and R.L. Rosenberg, 'An Overview of MULTIBASE,' in Distributed Databases, (H.J. Schneider, ed.), North Holland Publishing Company, 1982, pp. 153-184.

[RBFG80]
Rothnie, P.A., P.A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. Landers, C. Reeve,

D.W. Shipman, and E. Wong, 'Introduction to a System for Distributed Databases (SDD-1),' ACM Trans. on Database Systems Vol. 5, No. 1, March 1980, pp. 1-17.

[SA80]
Selinger, P.G., and M. Adiba, 'Access Path Selection in Distributed Database Management Systems,' Proc. International Conference on Databases, University of Aberdeen, Aberdeen, Scotland, July 1980.

[Ship81]
Shipman, D.W., 'The Functional Data Model and the Data Language DAPLEX,' ACM Trans. on Database Systems Vol. 6, No. 1, March 1981, pp. 140-173.

[SBDG81]
Smith, J.M., P.A. Bernstein, U. Dayal, N. Goodman, T.A. Landers, W.-T.K. Lin, and E. Wong, 'MULTIBASE -- Integrating Heterogeneous Distributed Database Systems,' Proc. AFIPS National Computer Conference, Vol. 50, 1981, pp. 487-499.

[SS77]
Smith, J.M., and D.C.P Smith, 'Data Base Abstractions: Aggregation and Generalization,' ACM Trans. on Database Systems, Vol. 2, No. 2, June 1977, pp. 105-133.

[Ston77]
Stonebraker, M., 'A Distributed Database Version of INGRES,' Proc. Berkeley Workshop, May 1977.

[WDHL82]
Williams, R., D. Daniels, L. Haas, G. Lapis, B. Lindsay, P. Ng, R. Obermarck, P. Selinger, A. Walker, P. Wilms, R. Yost, 'R*: An Overview of the Architecture, Proc. Second Int'l. Conf. on Databases -- Improving Usability and Responsiveness, Jerusalem, Israel, June 1982.

[YO79]
Yu, C.T., and M.Z. Oszoyoglu, 'An Algorithm for Tree-Query Membership of a Distributed Query,' Proc. IEEE COMPSAC 79, November 1979, pp. 306-312.