Query Processing in a Distributed Environment

by

Han Ying Chao

B.S., Soochow University, Taipei, Taiwan R.O.C. 1980

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

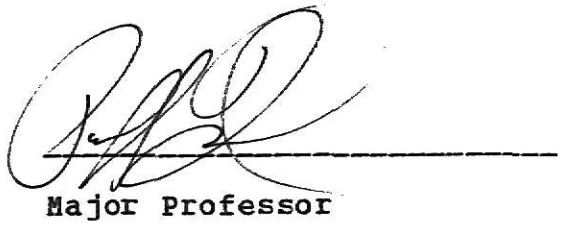
Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1982

Approved by:



Major Professor

Acknowledgements

The writer is indebted to her major advisor, Dr. Paul Fisher for his knowledge and perspectives in the development of this research.

Special gratitude goes to my parents, Mr. and Mrs. Chin Chi Chao, for their encouragement throughout my graduate studies. They instilled in me the value of education.

THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.

THIS IS AS RECEIVED FROM CUSTOMER.

Query Processing in a Distributed Environment

Table of Contents

Chapter 1 Introduction

Distributed computing systems are a new type of system. They emerged from the necessity to provide a vast amount of data to satisfy the requirements of geographically separate end users. Therefore, the need for data bases containing required information is becoming more and more a part of our society. Several approaches concerned with data access by means of a query language have been proposed. To achieve the goal of flexibility and efficiency when accessing data in a distributed environment, query processing needs to be studied carefully.

First, the diverse categories of abstract query languages are discussed as they relate to the relational data model. The form and structure of each language are overviewed. Subsequently, several current query languages are discussed with respect to the type of the abstract query language used.

The purpose of this report is to focus on the procedure for analyzing, decomposing, transmitting, and synthesizing a query. The object is to try to obtain the optimal solution of query processing under the conditions of minimum network traffic and minimum response time. Some methodologies dealing with query processing in both homogeneous and heterogeneous environments are discussed. These include the network decomposition algorithm, query processing mechanisms for heterogeneous DDB's, multi-level

integration, and partial integration. Although the technique for the latter two methods is not practical to implement, it demonstrates future tendencies.

Chapter 2 Definition of Data Bases

In order to obtain an overall concept of diverse data bases, the hierarchical, network, and relational data bases will be discussed respectively.

Section 2-1.A Hierarchical

A data base with data structure that a parent record type may have one or more child record types satisfies the restrictions (1) a child record type may not have more than one parent record type, and (2) an M:N relationship of instances between two record types is not allowed.

The relationships between records (or segments as termed in hierarchical data base) are not explicitly expressed as those sets of network data model. The hierarchy of segments are ranked by means of the parent-child relationships. The root segment type is the one with the highest hierarchy. All other segment types, called dependent segment types or child record types, are arranged in the tree structure in 'physical' parent-child relationships. Currently, IBM's Information Management System (IMS) is one of the most commercially used systems based on hierachical data model.

Section 2-1.B Network

A hierarchical data base can be viewed as a special case of network data base. Here, the network data base can

be defined as a collection of one or more record types, in which any record type can be related either as a parent or as a child record type to any number of other record types; for each parent record occurrence there may be one or many related child record occurrences. Also, there are two restrictions that must be followed, 1) a record cannot be both an owner and a member of a set, and 2) the same record occurrence cannot be in more than one set occurrence.

One of the basic concepts dealing with the network data base is the set type. A set type may be defined as a named relationship between record types. There may be an arbitrary number of occurrences of a member record type in a set, but with only one occurrence of its owner record type. Rules for the formation of sets can be found in (Cardenas). TOTAL, developed by Cincom Systems Inc., which deals with network data base structures is one of the most used systems.

Section 2-1.C Relational

The following discussion of data base in the distributed environment adopts the notation of the relational data base. The relational data model was conceived by E. F. Codd (Codd). It was intended to provide a means for easily understanding and handling data by users with little or no training in programming, and no consideration of positional, pointer or access path aspects. In effect, the system is free to choose any physical

structure for storage of data. A relational data base is made up of a set of relations or flat tables (that is, without repeating groups), in which relationships can be expressed as two relations having a common value in one field or domain. The relationships can be either 1:N or M:N.

We now may define relation by means of mathematical set theory. Given a collection of sets D_1, D_2,..., D_n (not necessary distinct), R is a relation on these n sets provided that it is a set of ordered n tuples d_1, d_2,..., d_n such that d_1 belongs to D_1, d_2 belongs to D_2 ,..., d_n belongs to D_n. Sets D_1, D_2 ,..., D_n are the domains of R. The number n is called the degree of R, and the number of tuples in R is called its cardinality.

When discussing relations it is customary to represent a relation as a two-dimensioned table with each row representing a tuple. Several properties derived from the definition of relation can be given as follows:

1) no two rows are identical;
2) the ordering of rows is immaterial; and
3) the ordering of columns is significant.

Under these observations, the degree of the relation is the number of columns and the cardinality of the relation is the number of rows.

Section 2.2 Definition of Homogeneous DDB

Distributed Data Bases (DDB) basically arise from an increasing number of needs which include the requirements for faster, easier access to data for decision making purposes, as well as providing a highly reliable and secure system. In addition, the increasing geographic dispersion of the end users within an organization provides the essential driving force for the provision of such systems. Before the discussion of both Homogeneous and Heterogeneous DDB, the definition of DDB will be given first. Distributed Data Bases can be defined as a series of separate but interworking subsystems which have processing power or storage or both. Or according to Draffan and Poole (Draffan), they defined five components of a distributed data base management system: the data base, DBMS, user process, distributed executive (the network DBMS) and an adaptation module (the network access process). In this case, the set of data forming the distributed data base is stored on a number of nodes over the network which supports the whole system. Thus, the local data base is the subset of the distributed data base at a particular node. Figure 2.1 shows a complete node in a distributed data base

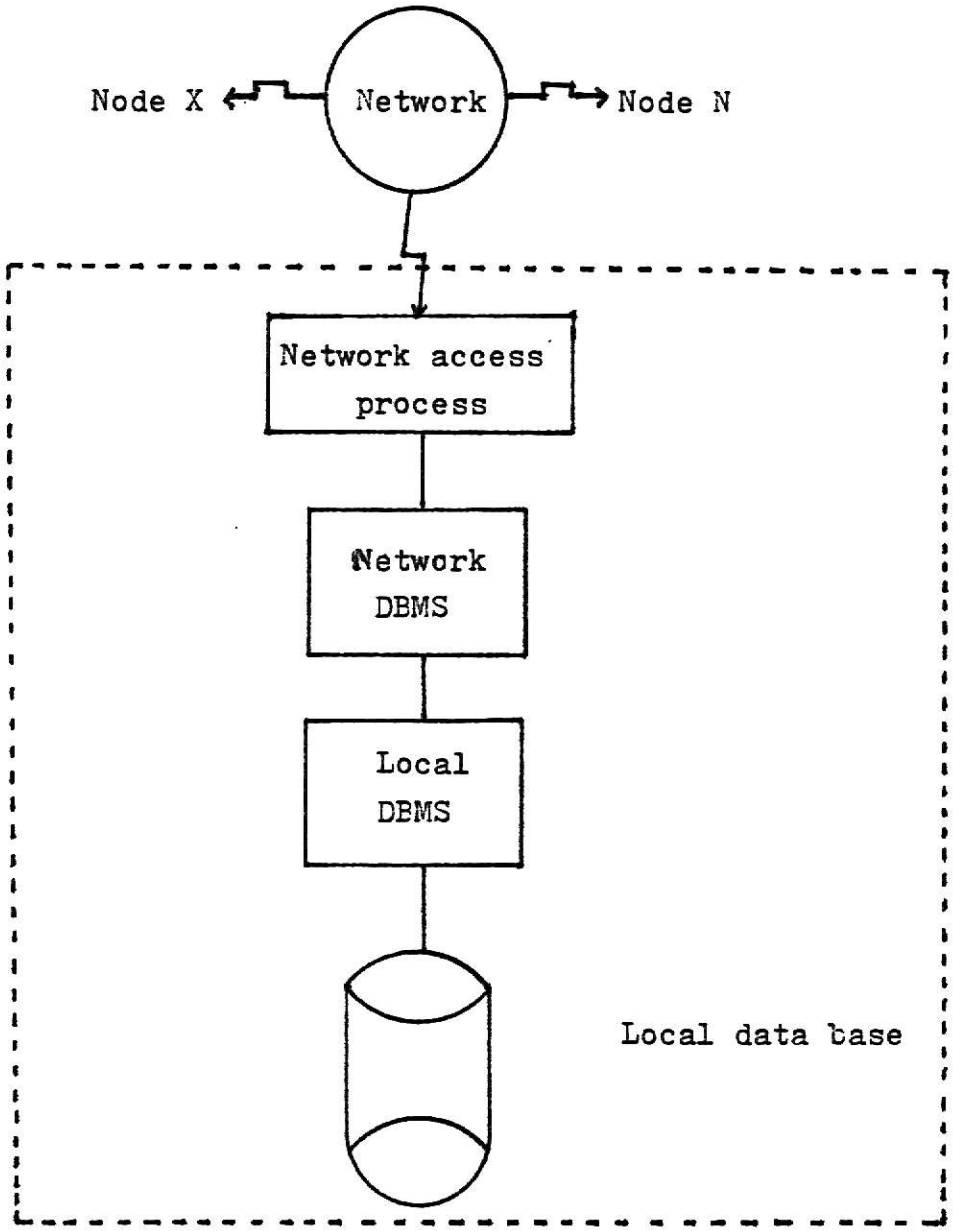


Figure 2.1 A complete node in a distributed data base.

environment.

The homogenecus DDB can be defined as 'a distributed data base where several identical data base management systems exist at sites of a computer network. The hardware of each site is also identical and permits each DBMS to co-operate in presenting to the user either a global or local external schema' (Draffan).

Several possible different criteria according to which two systems may be defined as homogeneous are included as follows:

(1) Use the same data model (hierarchical, network, relaticnal,...) to describe the data base. Further, this can be subdivided into a broad sense and a narrow sense. In the first case, it is required that data models be identical; in the second case, it is only required that both data models belong to the same data model family.

(2) Support the same data description language (DDL), or two equivalent DDLs having a few syntactical differences.

(3) Support the same data manipulation language (DML), or two equivalent DMLs having few syntactical difference.

(4) Provide for the same user service (offering the same functions), with either equal or different service level.

(5) Perform the same functions equal internal algorithms: for query processing, for consistency controls, for data storage, etc.

(6) Perform the same functions at equal cost.

(7) Implement (are programmed) in the same way (This criterion leads back to identical systems).

Section 2.3 Definition of Heterogeneous DDB

The heterogeneous DDB can be defined as 'a distributed data base where several dissimilar data base management systems exist at sites of a computer network. The hardware at each site may be identical in which case the software is dissimilar. Each DBMS may co-operate in presenting to the use either a global or local external schema. Co-operation is achieved by means of the use of a common data model to which each local DBMS maps' (Draffan). Or, the hetergeneous DDB can be defined as in a Distributed Data Base Management System (DDBMS) environment, where the local data base management systems are not identical.

Chapter 3 Query Language

This chapter is intended to discuss the three major approaches to the design of languages for expressing queries about relations. Query languages for the relational data model can be divided into two broad classes:

1. Algebraic languages, where queries are represented by applying spcialized operators to relations, and

2. Predicate calculus languages, where queries describe a desired set of tuples by specifying a predicate the tuples must satisfy.

The calculus-based languages can be further divided into twc categories, depending on whether the primitive objects are tuples or are elements of the domain of some attributes. Thus, we have in total three distinct types of query languages. All these languages mentioned above are aimed at avoiding the procedurality, physical access path dependency, and programming orientation of navigation languages (such as DL/1 and DML).

In this chapter, the relational algebra and the two forms of relational calculus, called tuple relational calculus and domain relational calculus will be discussed first. Then, one typical query language for each of the three types will be included: ISBL, an algebraic language; QUEL, a tuple calculus language; and Query-by-Example, a domain calculus language. Still, other languages which

intermediate between algebra and calculus, such as SQUARE and SEQUEL, will be included herein.

Section 3.1 Relational Algebra

Recall that a relation is a set of n-tuples for some fixed n, that is, the degree (or arity) of the relation. While defining relational algebra, it is assumed that columns need not be named, and the order in tuples is significant. Furthermore, let's assume that all relations are finite when dealing with relations of a data base.

The operands of relational algebra are either constant relations or variables denoting relations of a fixed degree. The degree associated with a variable will be mentioned only when it is necessary. Basically, a relational algebra operator uses one or more relations as its operand(s) and then produces a relation. There are five basic operations that serve to define relational algebra. Each of them will be discussed respectively.

1. Union. The union of relations R and S, denoted as R $\cup$ S, is the set of tuples that are in R or S or both. This operator can be applied with relations of the same degree.

2. Set difference. The difference of R and S, denoted as R - S, is the set of tuples in R but not in S. Here, R and S still are required to have the same degree.

3. Cartesian product. Suppose R and S are relations of degree n_1, and n_2, respectively. Then, the Cartesian product of R and S denoted as R x S, is the set with $(n_1 + n_2)$-tuples where the first n_1 components form a tuple in R and the last n_2 components form a tuple in S.

4. Projection. This is an operator for constructing a 'vertical' subset of relation, that is, a subset of the original relations with only the specified domains and with no duplicate tuples. If R is a relation of degree n, we let

$$\Pi_{i_1, i_2, \cdots, i_m}(R),$$

where i'js are distinct integers in the range 1 to n, denote the projection of R onto components $i_1, \ldots, i_m$, that is, the set of m-tuples $a_1\ a_2 \ldots a_m$ such that there is some n-tuple $b_1\ b_2 \ldots b_m$ in R for which $a_j = b_{i_j}$ for $j = 1,2,\ldots,m$.

5. Selection. Let F be a formula involving
   1) operands that are constants or component numbers,
   2) the arithematic comparison operators, $=$, $\neq$, $<$, $\leq$, $>$, and $\geq$, and
   3) the logical operators $\wedge$ (and), $\vee$ (or), and $\neg$ (not)

Then $\sigma_F(R)$ is the set of tuples t in R such that when for all i, the ith ccmponent of t for any occurrences of the number i in formula F is substituted, the formula F becomes true.

There are some additional algebraic operations such as intersection, quotient, join and natural join which will not be discussed here but can be found in (Ullman). All these additional algebraic operations can be expressed in terms of the five previously mentioned operations.

Section 3.2 Tuple Relational Calculus

Formulas in relational calculus may have the form of

$$\{ t \mid \psi(t) \} ,$$

where t is a tuple variable, that is, a variable denoting a tuple of some fixed length, and ψ is a formula built from atoms and a collection of operations.

The atoms of formula ψ may be the following three types.

1. R(s), where R is a relation name and s is a tuple variable. This atom represents the assertion that s is a tuple in relation R.

2. s[i] θ u[j], where both s and u are tuple variables and θ stands for an arithematic comparison operator. This atom represents the assertion that the ith component of s stands in relation θ to the jth component of u.

3. s[i] θ a or a θ s[i], where θ and s[i] are as in (2) above, and a is a constant. This represents the ith component of s stands in relation θ to the constant a.

At this stage, the notations of 'free' and 'bound' tuple variables will be introduced. The notion of a 'free variable' is analogous to that of a global variable in a programming language. A 'bound variable' can be defined as a variable with 'for all' or 'there exists'.

A tuple relational calculus expression can now be defined as an expression of the form $\{t|\psi(t)\}$, where t is the only free tuple variable in ψ. The relational algebra operators discussed above can be expressed by means of tuple relational calculus expression. This is shown in (Ullman).

Section 3.3 Domain Relational Calculus

The domain relational calculus is constructed by the same operators as the tuple relational calculus. Several essential differences are listed in the following:

1. There are domain variables to represent components of tuples instead of tuple variables.
2. An atom may be either of the form
   1) $R(x_1 x_2 \ldots x_n)$, where R is a n-ary relation and every x_i is a constant or domain variable, or

2) x θ y, where x and y are constants or domain variables and θ is a arithmetic relational operator.

3. Formulas in the domain relational calculus use the connectives $\wedge$, $\vee$, and $\neg$, as in the tuple relational calculus. Also, $(\exists x)$ and $(\forall x)$ are used to form expressions of the domain calculus, where x is the domain variable.

The notations of free and bound domain variables are defined in the domain calculus as in the tuple calculus. A domain calculus expression can be defined as of the form

$$\{ x_1, x_2, \cdots, x_n \mid \psi(x_1, x_2, \cdots, x_n) \},$$

where ψ is a formula whose only free domain variables are the distinct variables x_1, x_2, ..., x_n.

In order to conclude the discussion of three abstract languages which can serve as the part of a data manipulation language that extracts information from relations, one important feature will be specified here. Each of the three abstract query languages is equivalent in expressive power to the others, and they were proposed by Codd (Codd) to represent the minimum capability of any reasonable query language using the relational model. Therefore, they also serve as a benchmark for evaluating existing systems. In effect, almost all modern query languages are embedded within one of the notations mentioned above; some may be viewed as embedding a combination of these notations. A

language that can (at least) simulate tuple calculus, or equivalently, relational algebra or domain calculus, is said to be complete.

Section 3.4 ISBL

Information System Base Language (ISBL) is a query language generated by the IBM United Kingdom Scientific Center in Peterlee, England, for use in the Peterlee Relational Test Vehicle (PRTV) system. It closely approximates relational algebra. In both ISBL and relational algebra, R and S can be any relational expressions, and F is a Boolean formula. The correspondence of syntax is shown in Figure 3.1.

To print the value of an expression, have it proceeded by LIST. To assign the value of an expression E to a relation name R, we can delay the binding of relations to names in an expression until the left of the assignment. To delay evaluation of a name, have it proceeded by N!.

One simple example will be given here to illustrate the usage of ISBL. Suppose a relation exists in the data base as follows:

MEMBERS (NAME, ADDRESS, BALANCE)

Relational algebra	ISBL
$R \cup S$	R + S
$R - S$	R - S
$R \cap S$	R . S
$\sigma_F(R)$	R : F
$\pi_{A_1,\ldots,A_n}(R)$	R % $A_1,\ldots,A_n$
R |x| S	R * S

R |x| S stands for the natural join which can be found in (Ullman).

Figure 3.1 Correspndence between ISBL and relational algebra.

The query is required to find the names of members with negative balance. Then, in ISBL we can write the query as

LIST MEMBERS : BALANCE < 0% NAME.

Section 3.5 SQUARE and SEQUEL

SQUARE is a query language developed at the IBM in San Jose for the System-R DBMS. It has evolved into a language called SEQUEL, which is similar in concept to SQUARE, but the syntax is based on tuple relational calculus. Several important ideas about SQUARE will be discussed first.

In SQUARE, the union, difference, and intersection operators are expressed as in relational algebra. The Cartesian product of relations R and S can be expressed as:

$r \in R, s \in S$

Projection of relation R on attributed $A_1, A_2, \ldots, A_n$ can be expressed as:

$A_1, A_2, \ldots, A_n$ R

The way to accomplish the selection operator is by means of tuple calculus. Selection can be expressed in SQUARE as:

$r \in R : F'$

instead of $\sigma_F(R)$, where F' is F with r_A replacing the attribute A or the component number of that attribute in F.

In SQUARE, there is the capability to assign a computed relation to another relation name, by means of the assignment operator $\leftarrow$. An assignment can be expressed as

$$R_{A_1, A_2, \cdots, A_n} \leftarrow \langle \text{expression} \rangle$$

where the <expression> denotes an n-component relation, and the evaluated result of the expression will be assigned to relation name R, whose attributes are then named A_1, A_2,..., A_n. Assignment in SQUARE always results in immediate evaluation, unlike ISBL with feature of deferred evaluation.

The concept of mapping will be introduced here. Mapping, in effect, is a special kind of selection followed by a projection. The general form of mapping is

$$A_1, A_2, \cdots, A_n \; R \; B_1, B_2, \cdots, B_m \; (\theta_1 b_1, \theta_2 b_2, \cdots, \theta_m b_m)$$

where R is a relation name, the A's and B's are lists of attributes of R, θ_i is a comparison operator and is followed by a constant, b_i.

To conclude the discussion of SQUARE, one simple query will be provided. Assume that we have the relation schema MEMBERS on hand as was defined in the ISBL example. The query can be expressed as

$$_{\text{NAME}}\text{MEMBERS}_{\text{BALANCE}} \quad (<0)$$

to obtain the same result.

The reliance on subscripts in SQUARE's syntax causes some problems. Therefore, a syntax that places everything on a line will be defined. The major difference between SQUARE and SEQUEL is the usage of keywords to indicate the role of the relation and attribute names. In effect, the SEQUEL language is equivalent in power to SQUARE, but is

intended for users who are more comfortable with an English-keyword format than with mathematical notations of SQUARE.

The mapping can be expressed in SEQUEL as

SELECT $A_1, A_2, \ldots, A_n$

FROM R

WHERE $B_1 \theta_1 b_1 \ \ldots \ B_m \theta_m b_m$

SEQUEL presents the user with a consistent template for expression of simple queries. The user must specify the attributes he wishes to SELECT, the relation FROM which the query attributes are to be chosen, and the conditions WHERE the tuples are to be returned. In addition, a mapping may be applied to the results of inner mapping. This feature gives great flexibility in query expression. The formal syntax for SEQUEL queries can be found in (Chamberlin).

Now, let's consider again the query of relation MEMBERS. The same result can be obtained as in the earlier example by means of SEQUEL expression

```
SELECT NAME
FROM   MEMBERS
WHERE  BALANCE < 0
```

Section 3.6 QUEL

QUEL is a query language of INGRES, a relational DBMS developed at the University of California, Berkeley. The expressions are based on tuple calculus statements. The calculus statement has the form of

$$\{u^{(r)} \mid (\exists t_1)\cdots(\exists t_n)(R_1(t_1) \wedge \cdots \wedge R_n(t_n) \wedge u[1] = t_{i_1}[j_1] \wedge \cdots \wedge u[r] = t_{i_r}[j_r] \wedge \psi\}$$

where t_i is in R_i, u is composed of r particular components of the t_i's, and ψ is some condition to be satisfied. If ψ stands for any tuple calculus formula with no quantifiers, the above retrieve statement can be written in QUEL as

```
range of t1 is R1
        .
        .
        .
range of tn is Rn
retrieve (ti1 .A1 ,..., tir .Ar)
    where  ψ'
```

where A_m is the j_mth attribute of relation R_{i_m}, for m = 1, 2,...,n, and ψ' is the translation of qualification ψ into a QUEL expression.

Several steps have to be taken in order to perform the translation:

1. replace ψ's references to a component u[m] by a reference to $t_{i_m}[j_m]$. The term u[m] and $t_{i_m}[j_m]$ are equal.

2. replace any reference to $t_m[n]$ by $t_m.B$, where B is the nth attribute of relation R_m, for any n and m.

3. replace $\leq$ by <=, $\geq$ by >=, and $\neq$ by !=.

4. replace $\wedge$, $\vee$, $\neg$ by and, or, not, respectively.

To show the usage of QUEL, the same example will be presented here by QUEL.

```
range of t is MEMBERS
retrieve (t.NAME)
    wehere t.BALANCE < 0
```

Section 3.7 QBE

Query-by-Example (QBE) is a query language developed at IBM, Yorktown Hts. QBE is designed to be used sitting at a terminal, using a special screen editor to compose queries. It allows the user to call for one or more table skeletons to be displayed on the screen. The table skeleton is shown in Figure 3.2. The user then names the relations and attributes represented by the skeleton, making use of the screen editor.

Queries are executed by using domain vriables and constants, as in domain relational calculus, to form tuples that users assert are in one of the relations whose skeletons appear on the screen. Some of the variables, prefixed by their name with P., are printed. (All operators in QBE end with dot, and the dot is not itself an operator.) Whenever a tuple or combination of tuples matching the

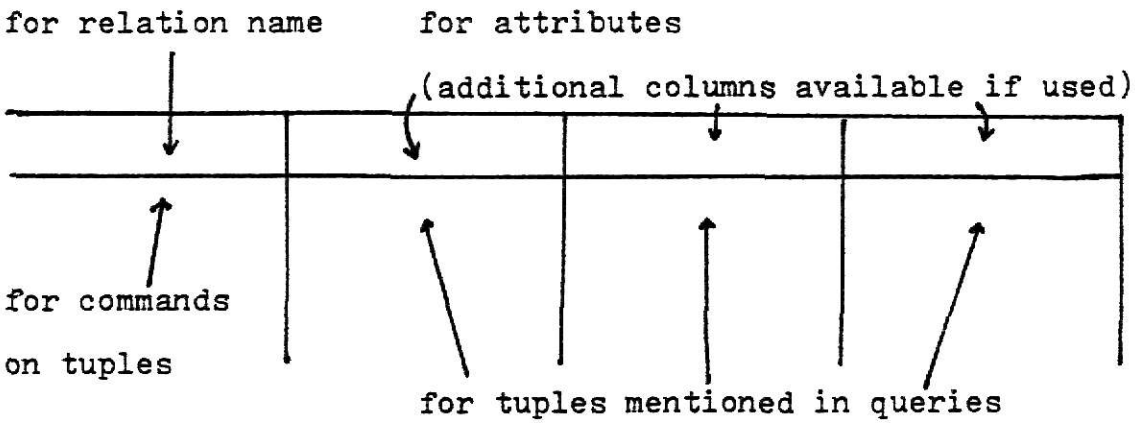


Figure 3.2 A QBE table skeleton

MEMBERS	NAME	ADDRESS	BALANCE
	_Oakes		_999

ORDERS	NAME	ITEM	QUANTITY
	_Oakes	_hotdog	_mucho

	P._Oakes	P._hotdog	P._mucho	P._999

Figure 3.3 Print names, items, quantities ordered, and balances.

conditions specified by the query are found, then the components for those attributes proceeded by P. are printed.

A large number of QBE queries correspond to domain calculus expressions of the form

$$\{a_1 a_2 \cdots a_n \mid (\exists b_1)(\exists b_2)\cdots(\exists b_m)(R_1(c_{11}, \cdots, c_{1n_1}) \wedge \cdots \wedge R_p(c_{p1}, \cdots, c_{pn_p}))\}$$

where each c_{ij} is an a_ℓ, a b_ℓ, or a constant, and each a_ℓ and b_ℓ appears at least once. In order to express any such query, we may display the table skeletons for all the relations named as $R_1, \ldots, R_p$, and create a variable name for each of the a's and b's. Then, for each term $R(c_{i1}, \ldots, c_{in_i})$ write a tuple in skeleton for R_i. If c_{ij} is a constant, place that constant in the jth component. If c_{ij} is one of the a's or b's, place the variable corresponding to that symbol there instead. Detailed discussion on this subject can be found in (Zloof).

An example is given here for illustrating the usage of QBE. In addition to the MEMBERS relation, we attach a new relation ORDERS as follows:

ORDERS (NAME, ITEM, QUANTITY)

Consider the problem: print the name, item, quantity and balance of the person named, for each order. In domain calculus, the query can be expressed as

$$\{a_1\ a_2\ a_3\ a_4 \mid (\exists b_1)\ (\text{MEMBERS}\ (a_1\ b_1\ a_4)\quad \text{ORDERS}\ (a_1\ a_2\ a_3))\}$$

The query is shown in Figure 3.3.

Chapter 4 Query Processing in Homogeneous DDB's

Query processing in a DDB corresponds to the translation of requests formulated in a high level language on one computer of the network, into a sequence of elementary instructions which retrieves data stored in the distributed data base. A possible diagram which specifies the architecture is shown in Figure 4.1.

In Figure 4.1, the upper part consists of the functions of supervision, distribution, and control of operations directly related to the distributed environment; the lower part corresponds to the functions of a local DBMS; both parts are linked by the communication network. In other words, the upper part of the software structure is used at the node where the query is generated for both analyzing the query and producing the optimal sequence of functional commands. Each command is further analyzed by the remote computer's local DBMS, then the optimal local data retrieval strategy is deduced. However, the execution of these commands may be postponed if there is a delay in retrieving data from another computer.

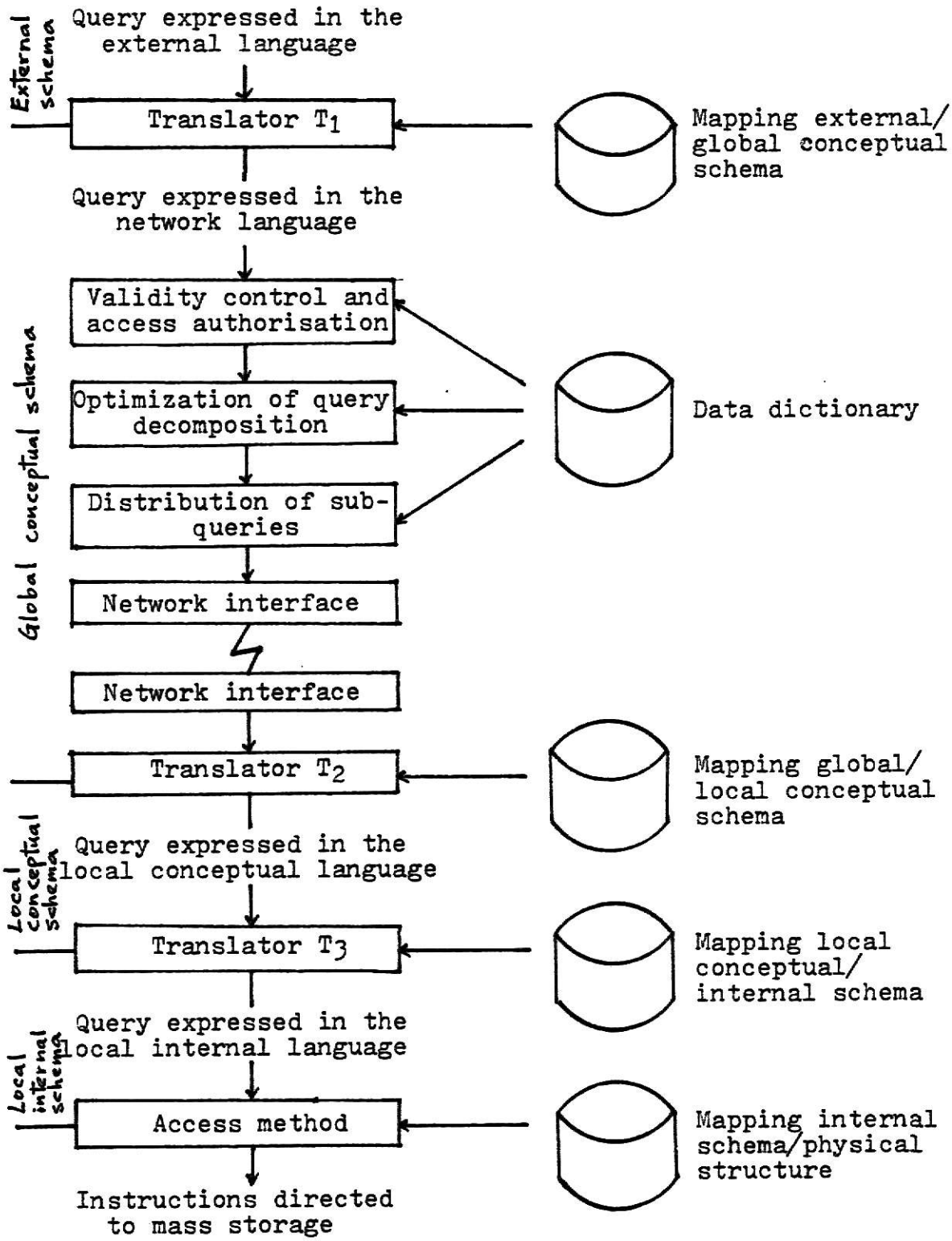


Figure 4.1 Global software architecture of a DDB.

Section 4.1 Multiple Machines in Homogeneous DDB Environment

In this section, the algorithms for processing data base commands that involve data in a homogeneous distributed data base environment consisting of multiple computer system each running the same operating system and data base server are discussed. In addition to the assumption that the data base consists of a collection of relations R_1, R_2,..., R_n, each relation, R_i, may be at a unique site or may be spread over several sites in a complete network.

A relation is 'local' if it is stored entirely at one site, and 'distributed' if portions of it are stored at different sites. At this point, it is assumed that relations are local, unless explicitly specified. The distributed data base can be extended over all or a subcollection of the sites in the computer network. The sites can be denoted a as S_1, S_2,$\cdots$,S_n. R_i^j means at site S_j there exists a fragment of R_i. Here, fragment means a subrelation, that is, a subset of the tuples, which can be denoted as a horizontal partition.

Two major types of communication networks are of concern, namely site-to-site and broadcast network. In the former one, let's assume that there is a fixed cost to send one byte of data from any site to any other site. In the latter one, it is assumed that the cost of sending data from one site to all sites is the same as that of sending the

same data from one site to a single other site. Regardless of which network model is used, let's assume that it is desirable to have bulk transmission (large volume data transmission).

The cost criteria considered are minimum response time and minimum communication traffic. In effect, these two criteria are not contradictory. For instance, an increase in network traffic will improve response time if it results in greater parallel processing.

Section 4.2 Network Decomposition Algorithm

The skeleton of the basic algorithm for decomposing a query will be presented first. The network decomposition algorithm has the following inputs:

1) the conjunctive normal form of the query (that is, the qualification composes clauses separated by AND's; each clause containing only OR and NOT).

2) the location of each fragment and its cardinality.

3) the network type (site-to-site or broadcast).

The algorithm is represented in Figure 4.2. The particular site where the query is generated is called the 'master' site. When processing a query, the master can communicate with one 'slave' at each site. These slaves are created by the master when appropriate. There are two kinds of commands that a master can send to a slave.

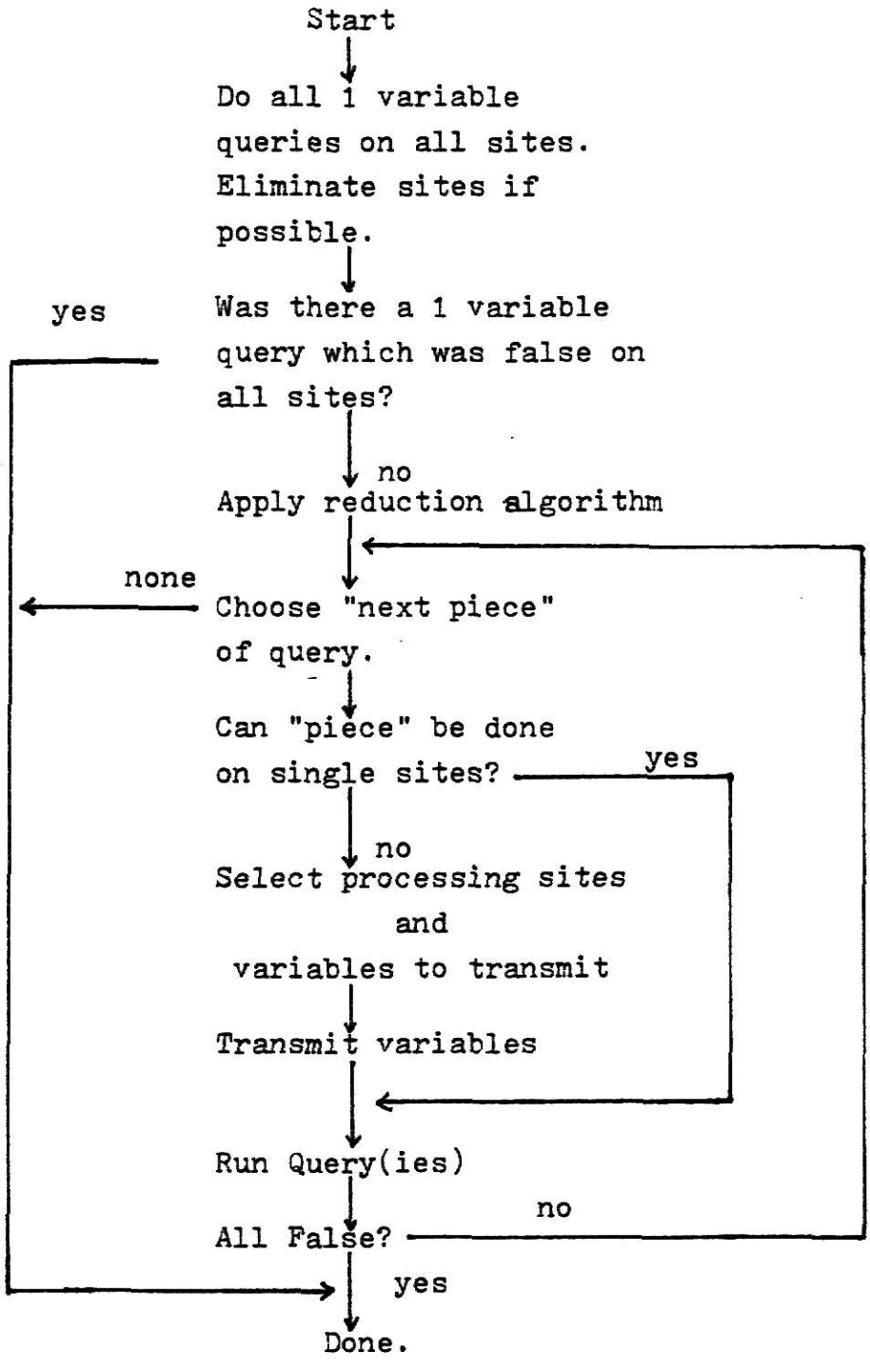


Figure 4.2 Network decomposition algorithm.

1) run the local query Q.

2) move the fragment R_i of relation R to a subset of the sites in the network, $S_1, S_2, \ldots, S_n$.

Section 4.3 The Optimization Problem

While processing each step in the algorithm, several steps should be studied carefully. The step 'choosing the next piece' will now be examined further. A query is broken into one or more parts and each part is processed in turn. The algorithm adopted here is called the 'greedy' algorithm (Draffan), that is, only a small set of alternatives is analyzed, choosing the optimum at each step without explicitly looking ahead to examine the global consequences.

For the moment, after all one variable subqueries have been removed, the reduction algorithm to transform the original query into its irreducible components will be applied.

$Q \rightarrow K_1, K_2, \ldots, K_i$

Since any component may span multiple sites, it may be desirable to subdivide the component into clauses.

$K = C_1, C_2, \ldots, C_j$

The question at hand is whether the entire component should be processed at once or subdivide it? The answer is to subdivide only if the condition --

size of the result relation from subdividing

< communication cost + processing cost

holds. In order to determine how to process a given subquery, it is necessary to minimize the following approximate cost function.

F(c1 * network-communication + c2 * processing-time)

Each part will be discussed separately.

Notationally, there are:

N total sites in the network

n relations referenced in the 'next piece'

We need to select:

K sites to be processing sites

R_p as the relation to be left fragmented

The processing sited can be numbered so that $1 \leq j \leq k$, and M can be defined as the number of sites where R_i has data stored. Therefore, at each processing site j, a query is concerned with

$Q = Q(R_1, R_2, \ldots, R_p^j, \ldots, R_n)$

To determine the communication cost given R_p and K, that is, a communication cost in which one relation R_p is left fragmented, and with k processing sites, the fragments of R_i have to be moved as follows:

for a processing site, R_i^j is moved to K - 1 other sites;

for a non-processing site, R_i^j is moved to K other sites;

and any fragments of R_p^j are moved to any one processing site.

The total communication cost can be defined as

$$comm = \sum_{j=1}^{K} C_{K-1}\left[\sum_{i \neq p} |R_i^j|\right] + \sum_{j=K+1}^{N} \left\{ C_K\left[\sum_{i \neq p} |R_i^j|\right] + C_1[|R_p^j|] \right\} \quad (4\text{-}1)$$

where C (x) denotes the cost of sending x bytes of data to K sites.

To estimate the relative processing time, proc(Q) is defined as the time required to process the query Q if it were done at a single site. The processing time at each site is given by

$$proc[Q_j] = \frac{|R_p^j|}{|R_p|} proc[Q]$$

and the overall processing time is

$$\max_j proc[Q_j] = \max_j \frac{|R_p^j|}{|R_p|} proc[Q]$$

under the assumption that all sites are of approximately equal processing speed.

To determine the cost of equalizing, the function pos(x) can be defined as:

pos(x) = x x > 0

pos(x) = 0 otherwise

The amount of data to be transmitted in equalizing the fragments of R_p is given by

$$\sum_{j=1}^{N} pos\left(|R_p^j| - \frac{1}{N}|R_p|\right)$$

This added network cost would result in a processing improvement from $\frac{|R_p^j|}{|R_p|}$ to $\frac{1}{K}$, under the assumption that each fragment of R was the same size.

At this point, let's assume that the overall optimization criteria exists solely to minimize communication costs. First, we will consider using a broadcast network and solving for K and R_p. For a broadcast network $C_K(x) = C_1(x)$ for all K = 1. By observing the formula 4-1, the first term will be zero if K = 1. The third term will be zero if every site which has part of R_p is a processing site. Therefore, K must be $\geq M_p$.

The rule for minimizing communication is given as follows:

If $\max_j \sum_i |R_i^j| > \max_i |R_i|$, choose K = 1 and choose the processing site to be the one containing the most data. No R_p exists in this case.

If $\max_j \sum_i |R_i^j| \leq \max_i |R_i|$, choose R_p to be the relation containing the most data and choose $K = M_p$. In other words, process the query at K sites.

To consider the site-to-site situation, we shall assume that $C_K(x) = K\, C_1(x)$. Independent of K, the optimal choice for R_p is the relation with the most data, that is, $\max_i |R_i|$. Once R_p is chosen, the value of K needed to minimize communication can be determined as follows: Suppose the sites are arranged in descending order of $\sum_i |R_i^j|$.

Then, choose K to be 1 if

$$\sum_{i \neq p} [|R_i| - |R_i'|] > |R_p'|$$

otherwise, choose K to be the largest j if

$$\sum_{i \neq p} [|R_i| - |R_i^j|] \leq |R_p^j|$$

The rule is as follows:

A site should be chosen as a processing site under the condition that the data to be received as a processing site is less than the additional data needed to send it as a nonprocessing site.

In the simplest environment, suppose only the communication cost is taken into account. The number of processing sites should be chosen according to the following rule:

Choose R_p to be the largest relation.

If N = 2 and n = 2 then K = 2.

If N = 2 then K = M_p for broadcast network

K = 1 for site-to-site network.

Finally, suppose the optimization criteria is concerned only with the objective of minimizing the processing time. In this case, we would like to have K = N so that we could distribute the work among all sites. At this point, still, it is necessary to choose one relation to remain fragmented. The actual processing time will be independent of whatever relation we choose. Thus, the R_p we are looking for should minimize network traffic, which can be expressed as:

$$\sum_{j=1}^{N} C_{N-1} [\sum_{i \neq p} |R_i^j|] + \sum_{j=1}^{N} pos [|R_p^j| - 1/N |R_p|]$$

Section 4.4 Summary

In order to obtain the optimal solution, several factors have to be considered in advance, including the precise relationships between processing time and communication costs, the distribution of data among N sites, and the true processing time for each site. The optimal solution for K should be in the range from 1 to N.

The problem for choosing R and K is still not solved. In this section, only two extreme cases are considered -- minimum network traffic and minimum processing time.

Chapter 5 Query Processing in Heterogeneous DDB's

The nodes within the network to support a distributed data base can be classified as storage, access, and exchange nodes, depending upon the function performed. Storage nodes are nodes which store those portions of data forming a DDB. They may be distinct nodes in a computer network. Access nodes are the places where nodes may interact with the DDBMS (Distributed Data Base Management System) to obtain access to the DDB. Finally, those nodes with both storage and access capability are called exchange nodes.

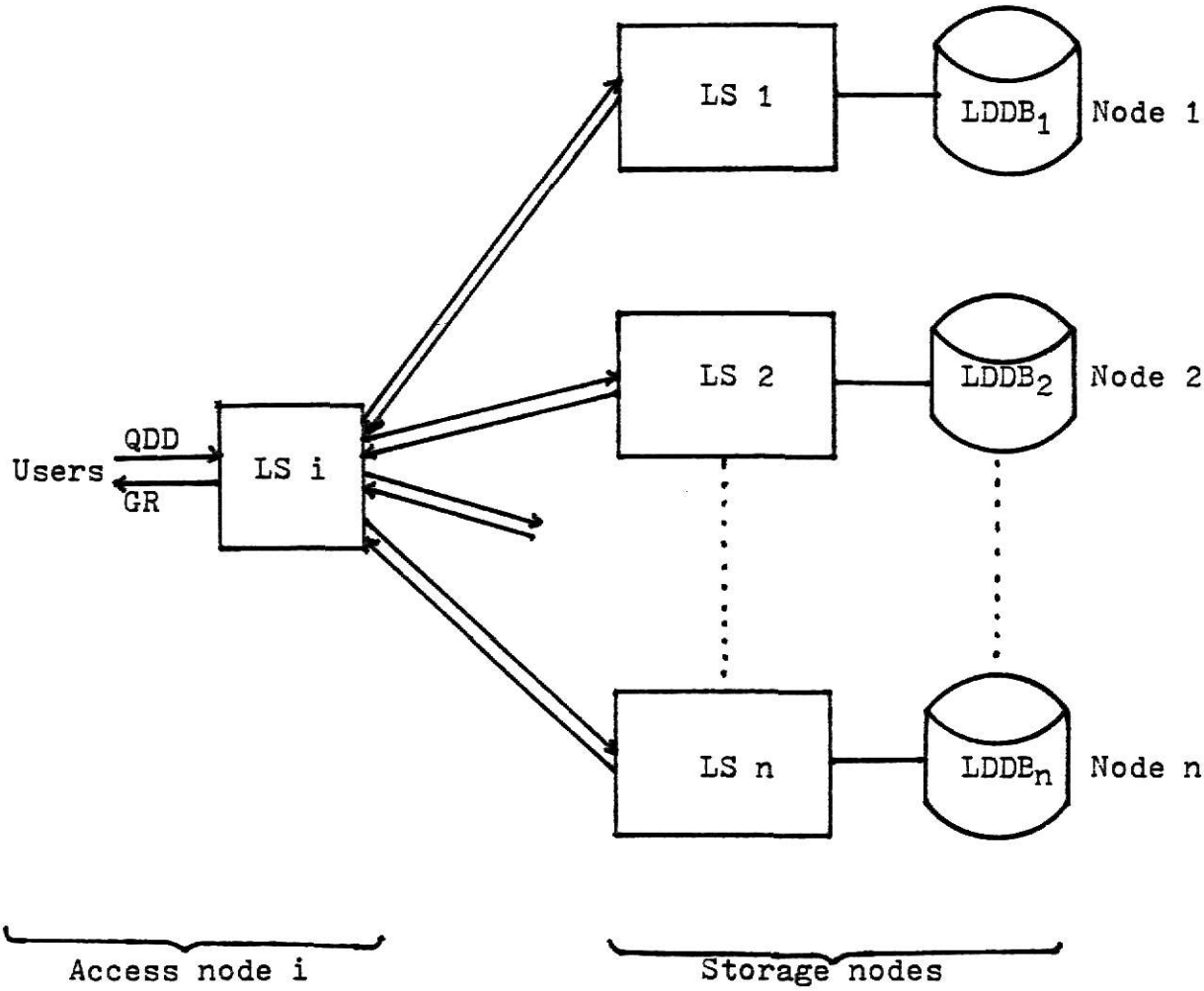
The subset of the DDB at a storage node can be defined as local DDB (LDDB). The DDB is the union of all LDDB's. At this point, it is assumed that the DDBMS is distributed over the nodes. The local system (LS) is that part of the DDBMS existing at each node. Thus, the DBMS is the union of all local LS's. The assumption made before implies a decentralized environment. It is also assumed that the LDDB is either required to form a data base itself or be part of a larger data base called the local data base (LDB). In addition, the local data management system is assumed to be a self-standing DBMS. It is called a local DBMS (LDBMS). Finally, the complement of the LDBMS within the local system is defined as the local distributed system (LDS).

Section 5.1 Query Processing Mechanism

Before the query processing mechanism is discussed in detail, one simplified example of query processing is given first. Systems perform query processing under the assumption that users are not required to know about data distribution. This problem will be discussed from two perspectives: 1) the access node, and 2) the storage node.

Users may issue queries for distributed data (QDD) at any of the access nodes (which include exchange nodes). Every query may involve a set of data stored over several storage nodes. The way for the local system to process such a query at the access node is to analyze the distribution of data required and then decompose the QDD into a set of sub-queries (called queries for local data or QLD). Then LS sends these QLD's to the local system of the corresponding storage node.

At the storage nodes, after the local system manipulates the QLD, the required data is sent back to the requesting access node. Finally, the access node synthesizes all local results to provide the user with the global response to his query. The overall mechanism which performs this function is summarized in Figure 5.1.



LS : local system

$LDDB_j$: local DDB at node j

QDD : query for distributed data

QLD_j : query for data local to node j

LR_j : local response to QLD_j

GR : global response to QDD

Figure 5.1 Simplified schema of an user access to the DDB.

Now, let's assume there are several different LDBMS's within the DDB envivonment. After decomposition of the user's QDD into sub-parts according to data distribution, each sub-part has to be expressed as a QLD based on the particular interface provided by the corresponding LDBMS. The process for transforming a decomposed QDD into a set of QLD's could be performed by LS at two places, either the access node or the storage node. It has been shown that to perform the process at the storage node is optimal for efficiency reasons (Draffan).

However, it is possible for different access node LS's to express queries using different languages. This condition should be avoided. For economical reasons, pivot languages and pivot data models are proposed. The former indicates a language common to all nodes and used for communication among different local systems; the latter refers to the data model corresponding to the pivot language. Figure 5.2 summarizes the method mentioned above.

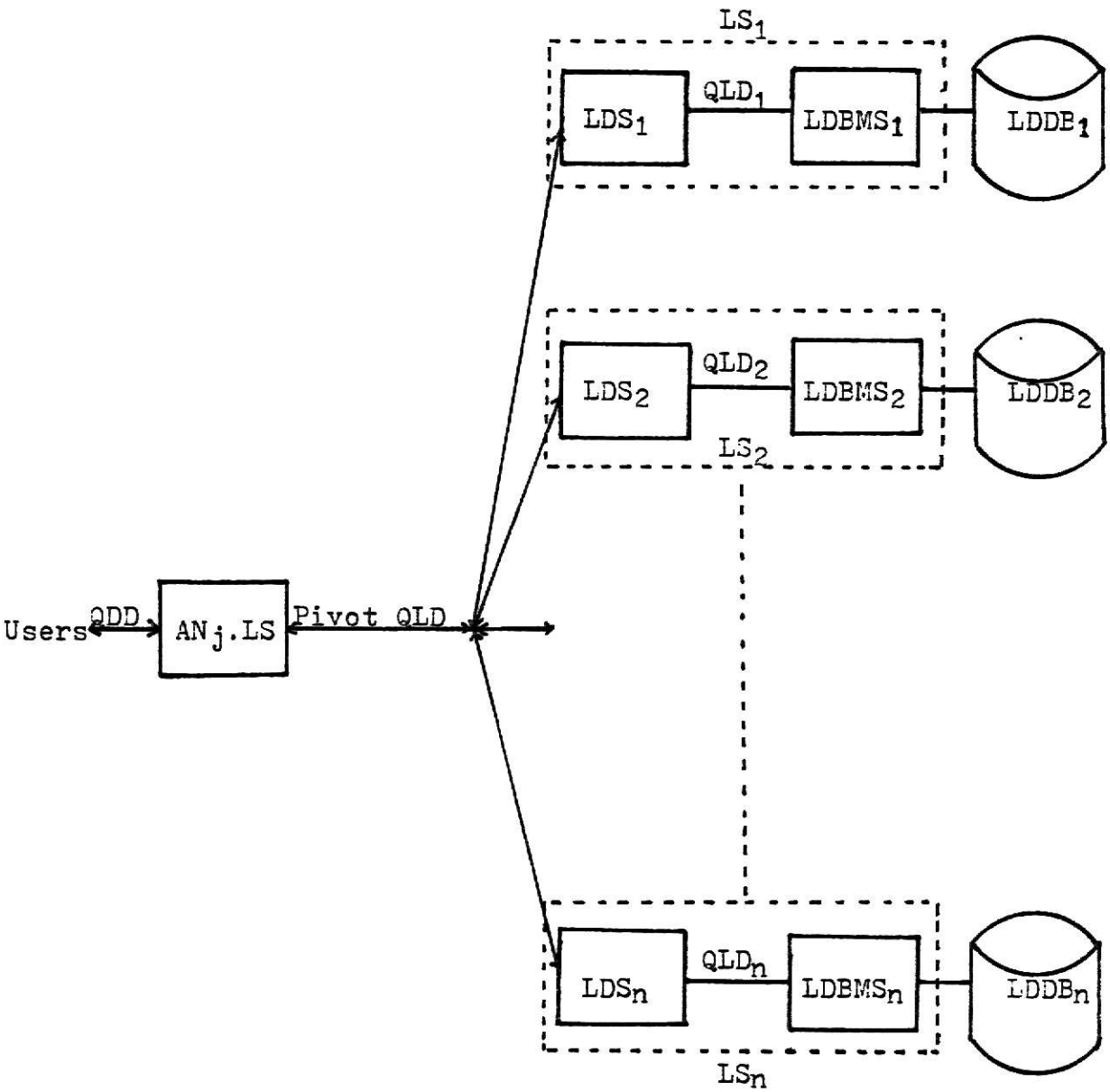


Figure 5.2 Heterogeneous DDBMS.

In addition, it is still possible for heterogeneous DDBMS's to design a multi-step structure which allows for a gradual translation of a pivot query into a local query. In order to do this, two steps will be taken at this point. First, an intermediate level will be introduced between pivot and local point. Local systems at access nodes would interface, using the pivot QLD, with a set of intermediate systems corresponding to a class of local systems (the class of relational systems, the class of network systems). Next, the intermediate system will initiate dialogue with the precise local system involved in the query. This approach is illustrated in Figure 5.3.

Section 5.2 Alternative Solutions

Section 5.2.1 Multi-level Integration

Alternatively, the LDBMS can be treated as a set of subsystems instead of an atomic element. To accomplish this, it is necessary to propose a set of communication levels, that is, a set of pivot languages at different system levels. Consequently, this will lead to a totally different approach. This plurality may offer flexibility and efficiency if the most suitable level for communication can be found for each function.

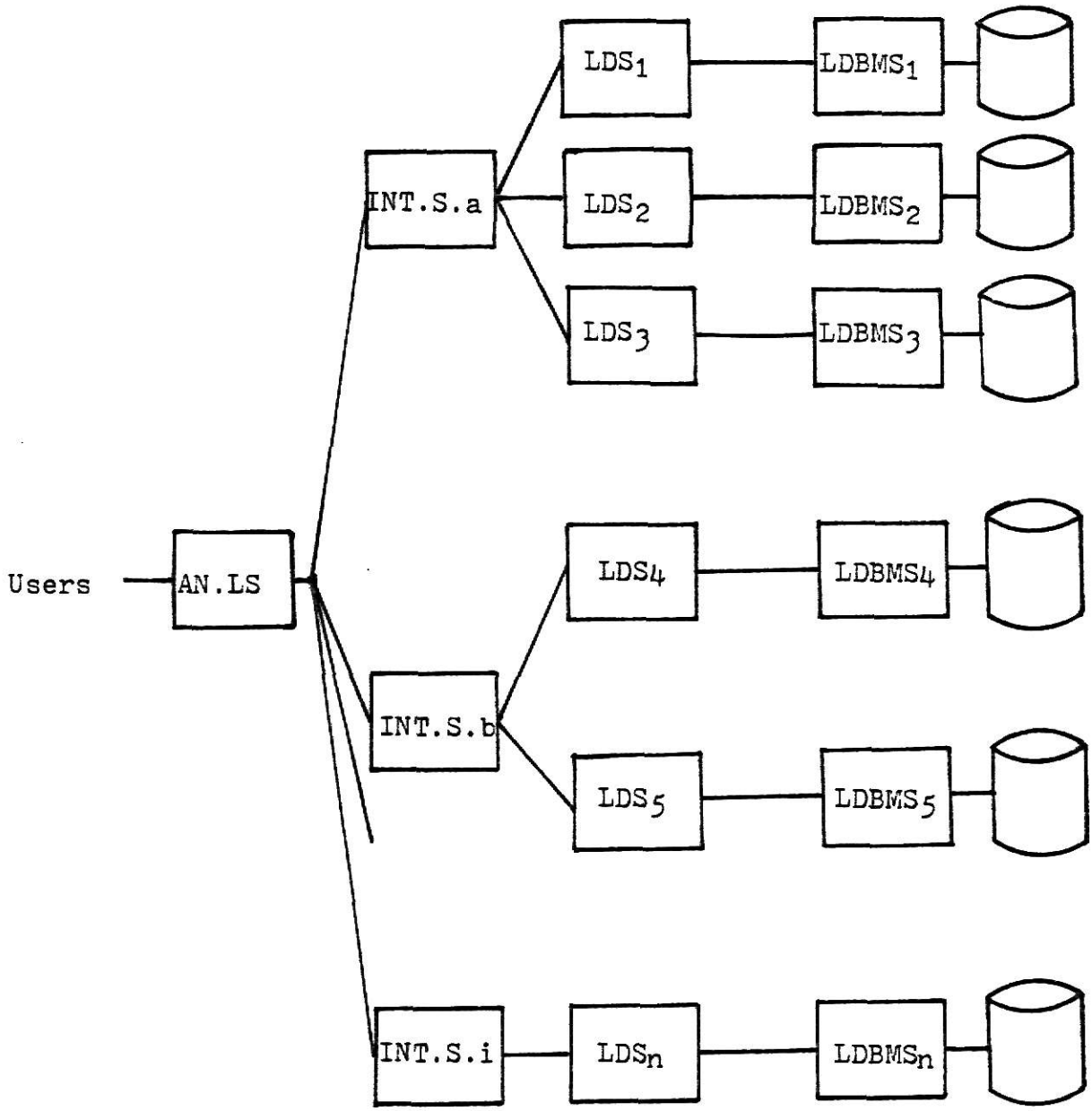


Figure 5.3 An example of heterogeneous DDBMS using intermediate systems. INT.a,b,...,i correspond to a class of homogeneous LDBMSs. $LDS_{1,2,...,n}$ correspond to a given local DBMS.

As proposed by (Spac), four system levels are defined: interactive query level, host language level, access method level, and microscopic level. However, in practice, only the upper levels (excluding the microscopic level) are implemented for monotype DDBMS's (the systems with identical local systems).

Section 5.2.2 Partial Integration

The integration of the LDBMS's into a standard local system performs only part of the functions performed by the DDBMS. This is defined as partial integration. The LDBMS's offer significantly different functions. For instance, within the query processing aspects, some LDBMS's may be able to force an order in the selected set of data, while others may not. In this case, total integration is not possible owing to the unreasonably high cost.

To solve this problem, alternative approaches may be used to provide a plurality of pivot languages or one pivot language with the capability of associating to a QLD with information corresponding to the additional needs of a given local system. Details for partial integration can be found in (Draffan).

Section 5.3 Summary

In this chapter, design issues for query processing based on a heterogeneous environment have been analyzed. The need for a heterogeneous user interface to the DDBMS has

been emphasized. Alternative solutions such as partial integration and multi-level integration need to be studied in the future. Until now, the current systems under development belong to the monotype or homogenous classes, but will extend to heterogeneous classes in the future.

Chapter 6 Conclusions

The purpose of this project was to examine the design of query languages based on a relational data model, and further, to provide a means for query processing in both homogeneous and heterogeneous distributed environments. To pave the way for discussion of query expression about relations, Chapter 3 specified three abstract query languages which can be used as standards for evaluating existing systems. In addition, the structure and form of query languages based on the notations mentioned above was examined.

According to Fisher, Schmidt, and Slonim (Fisher), it is a trend for partitioned systems to become realizable, attractive replacements for existing centralized systems. In effect, the advent of the distributed system causes severe problems when dealing with data access toward the data base. Chapters 4 and 5, presented several algorithms to provide guidelines for query processing in the distributed environment. The methodology proposed was independent of language expression such that a general concept was given to obtain the optimal solution.

The network decomposition algorithm discussed in Chapter 4 basically tries to move only the smallest amount of data and thus attempts to achieve the maximum amount of parallel processing possible. Heterogeneous DDBS's are still under development, and the way to process queries in

such environments is still an open question. It is hoped that the preceding pages have highlighted the main points concerning query processing in a DDB environment. It is further hoped that this report will provide the framework for future research in this area.

REFERENCES CONSULTED

1. Cardenas, A. F., Data Base Management Systems, Allyn, 1978.

2. Chamberlin, D. D.; and Boyce, R. F., 'SEQUEL: A Structured English Query Language', ACM-SIGMOD Workshop on Data Description, Access and Control (1974 : Ann Arbor, MI), P.249-264.

3. Codd, E. F., 'A Data Base Sublanguage Based on the Relational Calculus', Proceedings, 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, San Diego, California, November, 1971.

4. Codd, E. F., 'Relational Completeness of Data Base Sublanguages', in Data Base Systems, edited by R. Rustin, Vol.6 of the Courant Computer Science Symposia, Prentice-Hall, N. J., 1972.

5. Date, E. J., An Introduction to Databases Systems, 3rd ed., IBM programming series, 1981.

6. Draffan, I. W., Distributed Data Bases, Cambridge University Press, New York, N. Y., 1980.

7. Epstein, R.; Stonebraker, M.; and Wong, E., 'Distributed Query Processing in a Relational Data Base System', Proc. ACM-SIGMOD Conference on Management of Data, (June, 1978), 169-180.

8. Fisher, P.; Schmidt, D.; and Slonim, J., 'Consideration for Determining the Degree of Centralization or Decentralization in the Computing Environment' , Department of Computer Science, Kansas State University, 1978.

9. Kim, W., 'Relational Database Systems'. Computing Surveys 11, 3 (September ,1979), 185-211.

10. Spaccapietra, S., 'Problématique de conception d'un système de gestion de bases de données réparties', Thèse d'état, Université Pierre et Marie Curie, Paris, November, 1978.

11. Wong, E.; and Youssefi, K., 'Decomposition - A Strategy for Query Processing', ACM Transactions on Database Systems 1, 3 (September, 1976), 223-241.

12. Ullman, J. F., Principles of Database Systems, Computer Science Press Inc. , 1980.

13. Zloof, M. M., 'Query By Example', Proc. NCC 44 (May 1975).

Query Processing in a Distributed Environment

by

Han Ying Chao

B.S., Soochow University, Taipei, Taiwan R.O.C. 1980

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

1982

This paper describes the means for query processing in both homogeneous and heterogeneous distributed data base systems.

The query expressions to be discussed can be divided into three abstract query languages:

1. relational algebra,
2. tuple relational calculus, and
3. domain relational calculus.

One typical query language for each of the three types mentioned above is examined.

The query processing algorithm for homogeneous and heterogeneous environments will be discussed in Chapter 4 and Chapter 5 respectively. The cost criteria concerned with a minimum communication cost and minimum response time is also discussed. The overall query processing architecture will be examined. Some algorithms to be further developed (such as multi-level integration and partial integration) are overviewed.