A Framework for the Study of Query Decomposition for Heterogeneous Distributed Database Management Systems

K. Triantis
C.S. Egyhazy

TR 87-31

A FRAMEWORK FOR THE STUDY OF QUERY DECOMPOSITION FOR HETEROGENEOUS DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

TR 87-31

K. Triantis[a,b]

C. J. Egyhazy[c]

October, 1987

FOOTNOTES FOR TITLE PAGE

[a] Affiliation: Virginia Tech
Northern Virginia Graduate Center
Department of Industrial Engineering
and Operations Research

[b] This work was completed in part by K. Triantis under support of the 1987 NASA/ASEE summer faculty internship program.

[c] Affiliation: Virginia Tech
Northern Virginia Graduate Center
Department of Computer Science

Prefered Addresses for Correspondence

Konstantinos Triantis
Virginia Tech
Northern Virginia Graduate Center
Department of Industrial Engineering and
Operations Research
2990 Telestar Court
Falls Church, VA 22042
Tel: (703) 698-6086

or

Csaba J. Egyhazy
Virginia Tech
Northern Virginia Graduate Center
Department of Computer Science
2990 Telestar Court
Falls Church, VA 22042
Tel: (703) 698-6021

ACKNOWLEDGMENTS

Valuable insights and comments related to this research were provided by Dr. Barry Jacobs, Dr. Milt Halem, Dr. Isadore Brodsky, Dr. John Welch, Shyam Salona and Bharat Bhasker, Naji Wakim, Steve Hu, Judy Hu and Irene Lee. Their help is greatly appreciated.

ABSTRACT

This paper presents a framework for the study of the query decomposition translation for heterogeneous record-oriented database management systems. This framework is based on the applied database logic representation of relational, hierarchical and network databases. The input to the query decomposition translation is the query graph which is derived from the complex to basic, external to conceptual and logical optimization translations. Once the query graph is obtained the objective of the query decomposition translation is to break up a query expressed in terms of the actual or conceptual databases into its component parts or subqueries and find a strategy indicating the sequence of primitive or fundamental operations and their corresponding processing sites in the network necessary to answer the query. The query processing strategy is usually chosen so as to satisfy some performance criterion such as response time reduction. The choice of a query processing strategy is contingent on the successful estimation of intermediate results after each primitive operation. The pre-query decomposition translation, the query decomposition translation and the size estimation issues are presented through an example based on the current implementation of the Distributed Access View Integration Database (DAVID) currently being built at NASA's Goddard Space Flight Center (GSFC).

INDEX TERMS

1) query decomposition translation
2) heterogeneous record-oriented distributed database management systems
3) distributed query processing algorithms
4) applied database logic
5) generalized standard query language

I. INTRODUCTION

The primary objective of this paper is to propose a generic framework for the study of query decomposition for heterogeneous distributed database management systems (HDDBMS). We define query processing in a heterogeneous distributed database management system as that system feature where the user has both local and global query processing capabilities over any database/file management system in the network. We assume in our presentation that the resident database management systems are based on one of the three most popular record oriented data models, i.e., the relational [11], the hierarchical [12,31] and the network [12,31][1].

The purpose of the query decomposition process is to break up a query expressed in terms of the actual or conceptual database in the network into its component parts or subqueries and find a strategy indicating the sequence of primitive or fundamental operations and their corresponding processing sites in the network necessary to answer the query. What is meant by primitive or fundamental operations is the set of system dependent built-in operations which define the manipulations which can be performed on the records/files of the distributed database. The query processing strategy is usually chosen so as to optimize or satisfy some performance criterion such as, for example, response time minimization or reduction. The phrases 'query optimization'

and 'best strategy' indicate that the strategy selected by a query optimization algorithm is that for which the global optimum with respect to some performance criterion such as, query processing cost minimization, is attained. Since the problem of query decomposition is in general computationally intractable or NP hard [27] and is further hampered by the lack of precise statistical information about attribute values in the database, the term 'query optimization' is misleading. Especially since most strategies proposed are based on heuristic procedures. For these strategies, good but not necessarily globally optimum solutions with respect to some performance criterion are attained.

In the past, most of the research work in query decomposition has been carried out for homogeneous distributed relational database systems. The first algorithm for query decomposition for distributed relational database systems [33] translated the query into a sequence of move and process operations. This algorithm moved all the relations referenced by the query to an initial site, selected apriori, for processing. The final distribution strategy was obtained by improving upon this initial solution with a sequence of move and process commands until the lower cost sequence was found. A number of refinements of this algorithm followed, notably those which avoided the need for an initial apriori processing site and enhanced both the basic algorithm and the cost estimation techniques [1,4,10,18].

Most of the algorithms which followed, for example those reported by Hevner and Yao [18] and Bernstein, et al. [4], emphasize the minimization of transmission costs. More recently, Ceri and Gottlob [6] study how to optimize the sequencing of joins for a distributed relational DBMS for which horizontal fragmentation is assumed. They propose a method which includes a semantic optimization on joins, a semi-join reduction strategy and an integer programming formulation which finds the physical sequence of joins in the distributed network which minimizes the network transmission cost. Chu and Hurley [10], Yu and Chang [34] and Egyhazy [14] among others, describe methods for generating the query processing policy by considering both transmission and processing costs.

Queries can be first partitioned into subqueries, and these can be represented by a query graph. Their serial and parallel relationships and processing sequences are then represented by a query tree. For a single query graph, based on permutability properties, a set of equivalent query trees can be generated as shown by Smith and Chang [27]. One approach which selects the best query tree is based on the maximum temporary relation size reduction criterion [14]. A recursive algorithm which traverses the query tree by computing and estimating temporary relation size is proposed by Chao and Egyhazy [8]. Subsequently, Egyhazy and Triantis [15] proposed a heuristic based algorithm which also computes and estimates temporary relation sizes and determines

the sequence of relational operators and their corresponding processing sites. They later modified the heuristic based algorithm by incorporating an integer programming formulation which determines the relations in the network to be joined and their respective processing sites so as to minimize the sum of the network transmission and local processing costs [30]. This approach has been implemented by Stoler [28] using PASCAL.

A number of algorithms which determine a low cost sequence of semijoins have also been proposed. Bernstein and Chiu [3] were the first to introduce the idea of the semijoin operation as a way of solving tree queries[2]. The value of the semijoin operator is that it can reduce the amount of effort required to do a later expensive join, while the semijoin itself is often considerably cheaper. More recently, Chin-Wan and Irani [9] developed a heuristic algorithm which determines a low cost sequence of semijoins supported by a method which estimates the intermediate resulting relation sizes.

The query decomposition process for heterogeneous distributed database environments has not been researched extensively. According to Gliger and Luckenbaugh [17], the decomposition process in principle does not differ whether it applies to heterogeneous or homogeneous database systems. However, query decomposition algorithms for heterogeneous distributed databases are dependent in part on the specific data

representation of the various views which are defined by each local DBMS. The decomposition algorithms for HDDBMS are also contingent upon the definition and implementation of the primitive operations performed on the records/files.

One of the first and best known HDDBMS is Multibase [24], a prototype developed at Computer Corporation of America. Multibase translates the global query into the smallest possible set of single site subqueries. Each site may process several subqueries. All partial results are then collected and merged at a single site. The final result is subsequently sent to the query originating site. The Distributed Database Testbed System (DDTS) built by Honeywell Corporation and reported in Dwyer [13] is another early HDDBMS prototype. Its query processing approach is based on the work of Hevner for distributed relational DBMSs [18]. Also, the Integrated Manufacturing Database Administration System (IMDAS) for the Automated Manufacturing Research Facility (AMRF) at the National Bureau of Standards (NBS) [2] is currently being designed for heterogeneous distributed manufacturing environments. IMDAS is based on the data representation of the Semantic Association Model (SAM) [29]. There are a number of more recent and less known efforts made at building quasi HDDBMS prototypes, for example, the ISS at Arizona State University [19], and MRDSM at INRIA, France [25].

The framework proposed in this paper for the study of the query decomposition process for HDDBMS was inspired by the work of Jacobs [20,21]. This approach is being implemented in the DAVID (Distributed Access View Integration Database) system currently being built at NASA's Goddard Space Flight Center (GSFC).

The rest of the paper is organized as follows. A sample query is used throughout the presentation to present all the relevant concepts and notation. Section II presents an example of a heterogeneous distributed database management environment. The unified representation of the relational, hierarchical and network data models is based on database logic [20]. The basic building block of this representation is the cluster data structure. The query translation process is based on this data structure and other data structures, such as the query graph and execution tree [21]. It is assumed in this presentation that the data is distributed across the network in a predefined way. This means that the data is stored in specific DBMSs at each node. The information on the data distribution is found in the database aadministrator's dictionary. Section III describes how the original query is translated in terms of the conceptual databases and how the query graph is generated. The query graph is the data structure representing the query before it is decomposed into a sequence of primitive operations. The transition of a query graph to a particular execution tree is presented in

Section IV. This transition is based on the criterion of maximizing the reduction of temporary cluster or subcluster sizes and is analogous to the criterion of maximizing the reduction of temporary relation sizes for the relational case. Section V summarizes and concludes by presenting a framework for identifying the most important factors when researching the query decomposition process for HDDBMSs.

II. CONCEPTS AND NOTATION

Let's assume there is one or more resident DBMS at each node in a network system of interconnected heterogeneous databases. The example given in Figure 1 consists of five nodes: Nodes 1 and 5 have student registration information running under relational DBMSs. Node 2 has information about registration, courses, instructors and students running under a hierarchical DBMS, while node 3 has faculty information running under a network DBMS. At node 4 student information is stored as a hierarchical DBMS. This environment is depicted in Figure 1.

The unified representation of heterogeneous record oriented data models and the underlying canonical data structure of the distributed DBMS is based on the work by Jacobs [20]. He models databases in terms of "tables" and "rows" in tables. This representation is called a cluster. It is comprised of a single table or multiple embedded tables. The cluster representation can be depicted as follows.

$$\begin{aligned} &N\#.CN(\#)\ (T_1(a_1,\ldots,T_2(a_j,\ldots),..,T_e(a_b,\ldots),\ldots)),\\ &\qquad T_f(a_e,\ldots,(T_g(a_m,\ldots),\ldots)),\ldots,T_o(a_n,\ldots)\}\ ANAME \end{aligned} \tag{1}$$

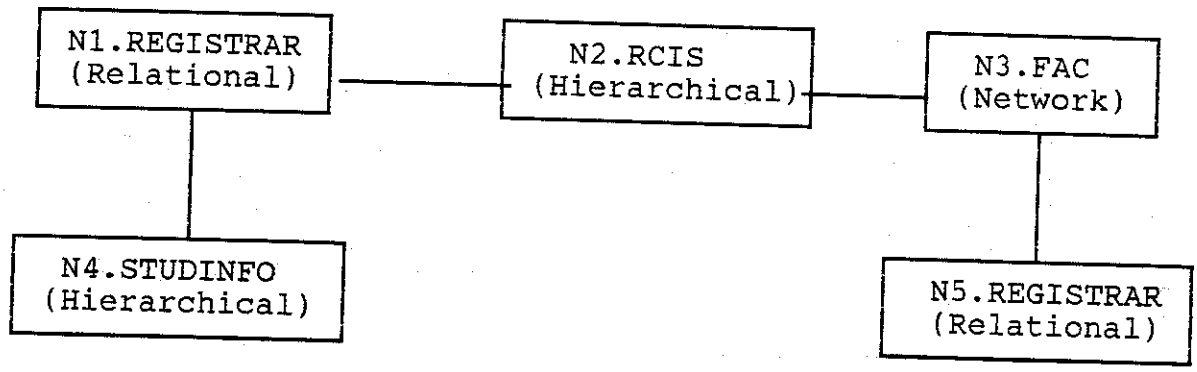

Figure 1. A Sample Heterogeneous Distributed Database Management System

Where, N# is the node number, CN(#) is the cluster name and its instance number and ANAME is the alias name for the cluster CN(#). T_i is the i^{th} table of the cluster CN and a_j is the j^{th} attribute name of the table T_i. For each cluster definition one can have more than one instance of this cluster. This depends on how many different sets of data values are used for the same cluster definition.

Using this notation, the cluster representations of the distributed DBMSs of Figure 1 are as follows.

N1.REGISTRAR(1) {REG(semester,year, dept,course#,sec#,id#,
instruct,studid,studname,grade)} R1 (2)

N2.RCIS(1) {REG(semester,year,COURSE(dept,course#,sec#,
INSTRUCTOR(id#,instruct),STUDENT(studid,studname,
grade)))} RCIS (3)

N3.FAC(1) {SYSTEM (DEPARTMENT(dept,chair,office,FACULTY
(id#,name,salary,degree)),SENIORITY(acadrank,
FACULTY(id#,name,salary,degree)),FACULTY(id#,
name,salary,degree))} F (4)

N4.STUDINFO(1) {STUDENT(studid,studname,ADDRESS(streetno,street,
city,state,zipcode),birthdate,dateadm,CREDITS
(dept,nocredit),gpa,major,minor)} S (5)

```
N5.REGISTRAR(1) {REG(semester,year,dept,course#,sec, id#,instruct,
          studid,studname,grade)}  R5                              (6)
```

The data manipulation language which can be used to define and manipulate clusters is Generalized SQL(GSQL) [21]. GSQL is a generalization of relational standard SQL [26]. Since GSQL is easy to understand and is based on standard SQL, it will be used subsequently to present the main concepts of this paper. For example, the GSQL cluster definition for the hierarchical database residing at node 2 of Figure 1 is as folows:

```
Define Cluster N2.RCIS(1) RCIS
REG TABLE
     (SEMESTER  CHAR [6];
      YEAR      CHAR [4];
      COURSE TABLE)
     KEY SEMESTER, YEAR
     COURSE TABLE
      (DEPT     CHAR [10];
      COURSE#   CHAR [6];
      SEC#      CHAR [5];
      INSTRUCTOR TABLE;
      STUDENT TABLE)
      KEY: DEPT,COURSE#,SEC#
      INSTRUCTOR TABLE
      (ID#      CHAR [10];
      INSTRUCT CHAR [30])
      KEY  ID#
      STUDENT TABLE
      (STUDID  CHAR [10];
      STUDNAME CHAR [30];
      GRADE    CHAR [5])
      KEY: STUDID
STORE AS DBASE (IMS);                                              (7)
```

In this definition, the schema for the cluster RCIS and each table in this cluster is given as well as the keys and any additional functional dependencies for each table. The last

statement of the definition indicates how the cluster instance is to be represented, i.e., the cluster N2.RCIS(1) is to be stored in the IMS DBMS. The GSQL definitions for the remaining clusters of Figure 1 are found in Appendix 1.

The format of a basic GSQL query which is used subsequently is as follows:

```
SELECT <SCHEMA>
FROM   <SOURCE CLUSTER>
WHERE  <BOOLEAN CONDITION>
```
(8)

The user can query an existing cluster or create a new cluster using the above format. The schema definition represents how the result will be reported or stored. The from part of the GSQL query indicates the source cluster definitions. In the where part the user specifies boolean conditions. Jacobs [21] illustrates how a GSQL query can be represented in terms of database logic[3].

III. EXTERNAL TO CONCEPTUAL MAPPING

For the sample HDDBMS of Figure 1, it was assumed that the cluster N2.RCIS is hierarchical and maintained at node 2. Suppose that some users prefer to view the data stored as N2.RCIS in terms of a relational database. A way to achieve this is to build the relational database as an external view (outside view) of the underlying conceptual (actual) hierarchical database. This means that some users can pretend that their view is relational even though in reality it is hierarchical. The external-to conceptual mapping problem can be stated as follows: How can one precisely express the relationship between two databases so that one can be maintained as an external view of the second conceptual database. The external to conceptual mapping problem can be expressed in terms of database logic [20] or GSQL [21]. Brodsky [5], among others, has discussed its solution and implementation. The subsequent example query and the external to conceptual mapping problem is presented in GSQL. In addition, the database logic representation of the external to conceptual mapping problem for this example query is presented in Appendix 2.

Along with the GSQL definitions of the conceptual clusters outlined in section II, assume that the user has defined the following external databases.

```
CREATE EXTERNAL CLUSTER  N1.REGISTRAR(2) ER1
SELECT   * R1
FROM  N1.REGISTRAR(1) R1
      N4.STUDINFO(1) S
WHERE R1-REG.STUDID = S-STUDENT.STUDID
```
(9)

```
CREATE EXTERNAL CLUSTER N2.RCIS(2) ERCIS
SELECT  * RCIS
FROM     N2 RCIS(1) RCIS
         N3 FAC(1) F
WHERE RCIS-REG-COURSE-INSTRUCTOR.INSTRUCT =
          F-SYSTEM-FACULTY.NAME
```
(10)

The external clusters ER1 and ERCIS are relational and hierarchical external views of R1 and RCIS respectively. In the case of ER1, the user views only a subset of the data found in R1. The cluster rows viewed by the user are those for which the STUDID values of the REG table of R1 equal the STUDID values of the FACULTY table of S. Notice that in the WHERE clause of the GSQL command the table paths are identified so that each field is uniquely identified. For ERCIS, the user views only a subset of the data found in RCIS. Again, the user views only those cluster rows of RCIS for which the INSTRUCT values of the INSTRUCTOR table of RCIS equal the NAME values of the FACULTY table of F.

Suppose the user is at node N1 and expresses a query based on the external clusters ER1 and ERCIS. The query requests the list of course numbers and grades in each department for each student name and student id number from the ER1 database. Each student id number should represent a student who is found in the ER1 database and has received a B for courses taken in the MATH

department and is also found in the ERCIS database and has received a B for courses taken in the ART depatment. The data of the result database is to be stored as a hierarchical database. This query can be represented by the following GSQL command.

```
CREATE ACTUAL CLUSTER N1.INFO(1) INF
SELECT    (STUDID,STUDNAME,COURSES) AS STUDENT
          (DEPT,COURSE#,GRADE) AS COURSES
FROM   N1 REGISTRAR(2) ER1
WHERE  STUDID IN
       (SELECT STUDID
       FROM N1.REGISTRAR(2) ER1
       N2.RCIS(2) ERCIS
     WHERE ER1-REG.STUDID = ERCIS-REG-COURSE-STUDENT.
                                 STUDID
     AND   ER1-REG.DEPT = 'MATH'
     AND   ER1-REG.GRADE = 'B'
     AND   ERCIS-REG-COURSE.DEPT = 'ART'
     AND   ERCIS-REG-COURSE-STUDENT.GRADE = 'B'    (11)
```

This is a nested query since the appropriate STUDID values must be found first and subsequently used to find the student name, department, course #, and grade values requested by the user. Therefore, the first step of the query translation process is to generate two queries from the above nested query, i.e.,

```
CREATE ACTUAL CLUSTER N1.TEMP_FRESULT(1) FR
SELECT    STUDID AS RES
FROM      N1.REGISTRAR(2) ER1
          N2. RCIS(2) ERCIS
WHERE     ER1-REG.STUDID = ERCIS-REG-COURSE-
                           -STUDENT.STUDID
AND       ER1-REG.DEPT  = 'MATH'
AND       ER1-REG.GRADE = 'B'
AND       ERCIS-REG-COURSE.DEPT = 'ART'
AND       ERCIS-REG-COURSE-STUDENT.GRADE = 'B'     (12)

CREATE  ACTUAL CLUSTER N1.INFO(1) INF
SELECT  (STUDID,STUDNAME,COURSES) AS STUDID
        (DEPT,COURSE#, GRADE) AS COURSES
FROM    N1.REGISTRAR(2) ER1
WHERE   STUDID IN N1.TEMP_FRESULT(1)              (13)
```

For brevity of the subsequent presentation the subsequent translation will concentrate on the nested component of the original query expressed by the GSQL command of expression (12).

The next step in the translation process is to express the GSQL query of expression (12) in terms of the conceptual or actual databases. This means that the resultant cluster N1.TEMP_FRESULT(1) FR will be attained from the actual databases N1.REGISTRAR(1) R1, N4.STUDINFO(1) S, N2.RCIS(1) RCIS and N3.FAC(1) F. The relationships among the resultant, external and conceptual databases is depicted by Figure 2.

Let Q(CN1;..,CN2,...) be a GSQL query in which cluster instance CN1 is defined in terms of a set of cluster instances which contains cluster instance CN2. Similarly, Q(CN2;...,CN3,...) is a GSQL query in which cluster instance CN2 is expressed in terms of a set of cluster instances which contains CN3. The composition of Q(CN1;...,CN2,...) with Q(CN2;...,CN3,...) can be represented as follows [21]:

$$Q(CN1;...,CN3,...) = Q(CN1;...,CN2,...) \circ Q(CN2;...,CN3,...) \quad (14)$$

where; 1) The FROM clause of Q(CN1;...,CN3,...) is obtained from the FROM clause of Q(CN1;...,CN2,...) by replacing the cluster name CN2 by the cluster name CN3; 2) The SELECT clause of Q(CN1;...,CN3,...) is obtained from Q(CN1;...,CN2,...) by

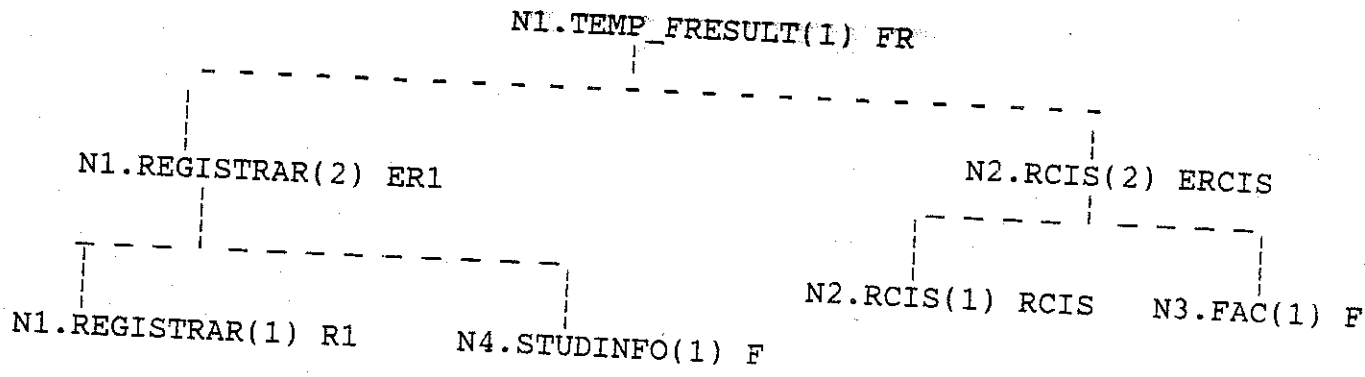

Figure 2. Relationships Among The Result External and Conceptual Databases

replacing the attributes from CN2 by the defined transformation functions of Q(CN2;...,CN3,...) applied to the corresponding attributes of Q(CN1;...,CN2,...); 3) The WHERE clause of Q(CN1;...,CN3,...) is obtained from Q(CN1;...,CN2,...) by replacing attributes from CN2 by the defined transformation functions of Q(CN2;...,CN3,...) applied to the corresponding attributes of Q(CN1;...,CN2,...).

Therefore, for our example, a possible first composition can be defined as:

```
Q(N1.TEMP_FRESULT(1); N1.REGISTRAR(1), N4.STUDINFO(1),
  N2.RCIS(2)) =
= Q(N1.TEMP-FRESULT(1); N1. REGISTRAR(2), N2.RCIS(2))o
  Q(N1.REGISTRAR(2); N1.REGISTRAR(1), N4.STUDINFO(1))      (15)
```

Following the steps outlined previously the GSQL query of expression (12) can be written as,

```
CREATE ACTUAL CLUSTER  N1.TEMP_FRESULT(1) FR
SELECT STUDID AS RES
FROM   N1.REGISTRAR(1) R1
       N4. STUDINFO(1) S
       N2. RCIS(2) ERCIS
WHERE  R1-REG.DEPT = 'MATH'
 AND   R1-REG.GRADE = 'B'
 AND   ERCIS-REG-COURSE.DEPT = 'ART'
 AND   ERCIS-REG-COURSE-STUDENT.GRADE = 'B'
 AND   R1-REG.STUDID = ERCIS-REG-COURSE-STUDENT.STUDID
 AND   R1-REG.STUDID = S-STUDENT.STUDID                   (16)
```

Lastly, N2.RCIS(2) is translated in terms of N2.RCIS(1) and N3.FAC(1). This gives the GSQL query of expression (13) in terms of the conceptual (actual) databases, i.e.,

```
Q(N1.TEMP_FRESULT(1); N1.REGISTRAR(1),N4.STUDINFO(1),
  N2.RCIS(1),N3.FAC(1)) =

=  Q(N1.TEMP_FRESULT(1); N1.REGISTRAR(1),N4.STUDINFO(1),
  N2.RCIS(2)) o Q(N2.RCIS(2); N2.RCIS(1), N3.FAC(1))                    (17)
```

Thus,

```
CREATE ACTUAL CLUSTER N1.TEMP_FRESULT(1) FRES
SELECT  STUDID AS RES
FROM    N1.REGISTRAR(1) R1
        N4.STUDINFO(1)  S
        N2.RCIS(1) RCIS
        N3.FAC(1) F
WHERE   R1-REG.DEPT = 'MATH'
 AND    R1-REG.GRADE = 'B'
 AND    RCIS-REG-COURSE.DEPT = 'ART'
 AND    RCIS-REG-COURSE-STUDENT.GRADE = 'B'
 AND    R1-REG.STUDID = RCIS-REG-COURSE-STUDENT. STUDID
 AND    R1-REG.STUDID = S-STUDENT.STUDID
 AND    RCIS-REG-COURSE-INSTRUCTOR.INSTRUCT =
          = F-SYSTEM-FACULTY.NAME                                       (18)
```

IV. QUERY DECOMPOSITION

At this point, the translation of the example query has resulted in accomplishing two tasks. First, the original complex (nested) query has been decomposed into two basic queries. Second, each of these basic queries has been translated in terms of the conceptual databases in the distributed network. The outcome of such a translation can be graphically depicted by a query graph. In our example, the basic query representing the nested component of the original query is depicted by the query graph of Figure 3. This query graph corresponds to the GSQL query given by expression (18).

The query graph of Figure 3 indicates that selections and semijoins are to be executed on four conceptual databases. Each conceptual database is represented by a node on the query graph. Attribute names for which selections are performed on clusters are represented by double circles. For example, the rows of table REG of cluster R1 for which GRADE = 'B' need to be selected. This information is depicted on the arc connecting the attribute value ('B') and the cluster node N1.REGISTRAR(1) R1. In general the arc notation depicting this information is given as follows:

Alias Cluster Name-Table1-Table2-
...-Tablen. Attribute Name = 'Attribute Value' (19)

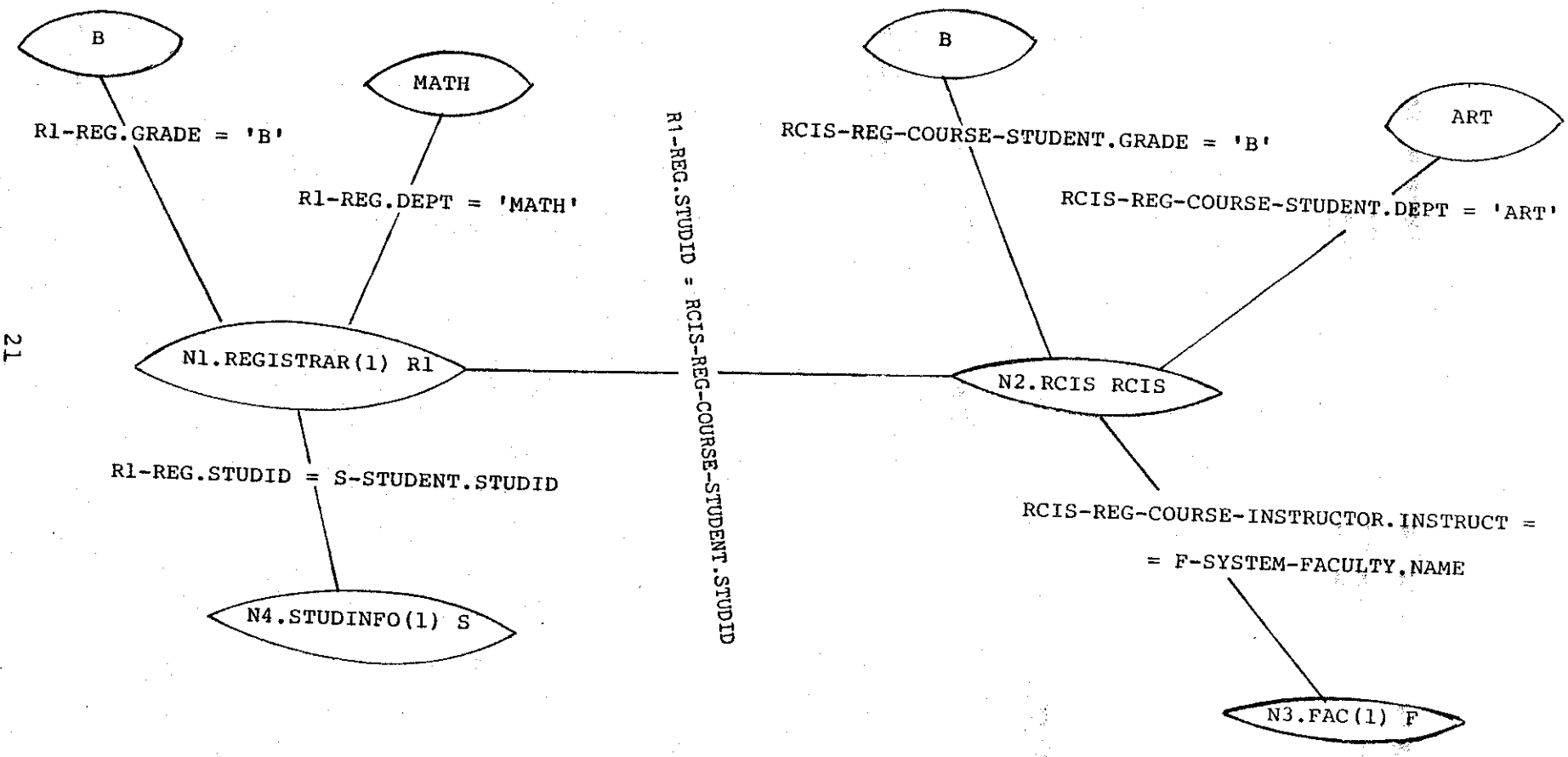

Figure 3. Query Graph of the Nested Component of the Original Query Expressed in Terms of the Conceptual Databases

This notation is important for clusters representing hierarchial and network databases since the attribute name is defined in terms of the table to which it belongs. Additionally, this table is defined in terms of a group of embedded tables represented by the subcluster row or table path. The arcs connecting cluster nodes define the joins or semijoins which need to be performed. For example, in Figure 3, N1.REGISTRAR is to be semijoined with N2.RCIS on STUDID. The notation for such join or semijoin operations is given as follows:

```
Alias Cluster Name1-Table1...-Tablen.Attribute Name1 =
    = Alias Cluster Name2-Table1-...-Tablem.Attribute
      Name2                                              (20)
```

Usually the attribute names on which join or semijoin operations are performed are identical. However, this need not always be the case. For example, the clusters N2.RCIS(1) and N3.FAC(1) are to be joined or semijoined on attributes INSTRUCT and NAME. It is assumed that these attributes represent the same logical entity.

The objective of the query decomposition process is to take the representation of the query expressed by the query graph and find an execution strategy which answers the query. This execution strategy is expressed as a sequence of primitive operations. The primitive operations which are considered in this

paper are the select, project, semi-join, join, transfer and establish operations. These operations are described explicitly in Section IV.A. In Section IV.B. an execution strategy is presented which answers the query depicted by the query graph of Figure 3.

In most cases, there are a number of execution strategies which answer any particular query. It is important that an execution strategy be chosen which reduces the query processing costs (both local and network transmission costs) and/or the query processing response time. The selection of such a strategy is usually accomplished before the actual execution of the query takes place and is based on the capability of estimating (forecasting) the result temporary cluster sizes after each primitive operation. Since the magnitude of both the response time and processing costs depend primarily upon the result temporary cluster sizes, the effectiveness of the query decomposition process in responding to queries in a timely and cost effective fashion is contingent upon the accuracy of the cluster size estimation methodology. A discussion of cluster size estimation techniques for each of the primitive operations is presented in some detail in Section IV.C.

A. Primitive Cluster Operations

The discussion of this section is motivated by the current implementation of the DAVID project at NASA. Examples of how these primitives are used are presented in the ensuing discussion concerning the generation of the execution tree from the query graph.

The first operation is that of a projection query where the DBMS traverses a specific source cluster instance and projects out those attribute values specified by the result cluster schema definition from every table row of each source cluster row. These values are then placed in a cluster according to the result cluster schema definition. The general format of the GSQL query for a projection operation is given as follows:

```
CREATE ACTUAL CLUSTER < N1 CN(i) ANAME>
SELECT <SCHEMA1>
FROM    <N1.CN'(j) ANAME'>                    (21)
```

In this case, the cluster instance N1.CN(i) ANAME is created and contains the data which are projected. Let's denote the schema of the result cluster N1.CN(i) as SCHEMA1. SCHEMA1 is defined in the SELECT statement. The data are projected from N1.CN'(j) ANAME', which is the source cluster. Notice that this is a local operation, since both the source and result clusters reside at

node N1. Also, it is possible for the source and result clusters to represent different types of databases. For example, our source cluster may represent a network database whereas, the result cluster a relational one.

The selection operation occurs when cluster rows are selected from a source cluster instance. For these cluster rows, specific attribute values satisfy certain boolean conditions. For this primitive operation the source and result clusters have the same cluster schema definitions. The GSQL query representing this operation is given as follows:

```
CREATE ACTUAL CLUSTER <N1.CN'(i) ANAME>
SELECT < * ANAME'>
FROM   < N1.CN'(j) ANAME')
WHERE  < BOOLEAN CONDITIONS>                    (22)
```

The cluster instance N1.CN'(i) is created and in this case contains the data for which the boolean conditions are satisfied.

The simultaneous selection projection operation occurs when cluster rows are selected from a source cluster instance. For these cluster rows, specific attribute values satisfy certain boolean conditions. Concurrent with the selection operation, is the projection of those attribute values specified by the result schema definition. The general format of the GSQL query for a selection projection operation is given as follows:

```
CREATE ACTUAL CLUSTER < N1.CN(i) ANAME>
SELECT <SCHEMA1>
FROM   <N1. CN'(j) ANAME'>
WHERE  <BOOLEAN CONDITIONS>                    (23)
```

The cluster instance N1.CN(i) ANAME is created and contains the data which satisfy the boolean condition specified by the WHERE statement and are projected according to the SCHEMA1 definition of the SELECT statement. The source cluster N1.CN'(j) ANAME' is specified by the FROM statement. Again, this is a local operation since both the result and source clusters reside at node N1.

The objective of the semijoin operation is to have the DBMS run through two cluster instances N1.CN(i) and N1.CN'(j) and select the cluster rows of N1.CN(i) which satisfy specific boolean conditions. The result of this operation is a cluster instance N1.CN(k) which is comprised of those cluster rows of N1.CN(i) which matched rows of N1.CN'(j) according to the boolean conditions. This match occurs for specific attribute values of each cluster row for both clusters N1.CN(i) and N1.CN'(j). These attributes are defined in the WHERE statement of the GSQL query.

In order to minimize the data manipulated by the semijoin operation, those attributes necessary to answer the query and to

perform the semijoin operation are projected from N1.CN'(j) ANAME'. The result cluster is defined as a table containing these essential attribute values. Thus,

```
CREATE ACTUAL CLUSTER <N1.CN'1(j) ANAME1'>
SELECT  <TABLE1>
FROM    <N1.CN'(j) ANAME'>                    (24)
```

Table1 contains the attribute values necessary to answer the query and those necessary to perform the semijoin operation as well as the virtual addresses of each table row of TABLE1. It is necessary to keep track of the table row addresses because they will subsequently become part of a map table which will be used to bring together all the attributes necessary to answer the query.

During the semijoin operation each cluster row of N1.CN(i) is traversed. The DAVID DBMS then goes through all the table rows of TABLE1 and searches for possible matches of specific attribute values for each cluster and table row. If a match occurs, the cluster row is inserted in the result cluster definition. Also, the virtual addresses of the inserted cluster row and of the matched table row are inserted in the MAP table. The MAP table is created since it facilitates the temporary representation of a cluster (TEMP) described in section IV.B. The TEMP representation has been created in order to facilitate

the manipulation of the data stored in the various clusters and the final assembly of data in order to obtain the result requested by the user. The general format of the GSQL query for a semijoin operation is given as follows:

```
CREATE ACTUAL CLUSTER  <N1.CN(k) ANAME1>
                       <N1.MAP(m) MAP1>
SELECT  <SCHEMA1>
        <SCHEMA2> AS <TABLE2>
FROM    <N1.CN(i) ANAME>
        <N1.CN1'(j) ANAME1'>
WHERE   <BOOLEAN CONDITIONS>                    (25)
```

In the above definition, SCHEMA1 represents the schema for both the source and result clusters N1.CN(i) and N1.CN(k), whereas, SCHEMA2 defines the schema of the map table N1.MAP(m). Also, N1.MAP(m) is comprised of the virtual addresses of the cluster rows of N1.CN(i) and N1.CN1'(j) for which the boolean conditions are satisfied. The implementation of a virtual or symbolic address is not trivial. Currently in the DAVID project the semijoin operation uses physical addresses. This creates however, serious difficulties in keeping track of the data especially when these data are transfered from one node to another.

Since in some cases the clusters which are to be semijoined reside in two different nodes in the distributed network, it is necessary to determine which one of the clusters is to be transferred across the network. If the objective is to minimize the amount of data transfer, the performance of the execution algorithm will depend mostly on the accuracy of the result temporary cluster size computations and/or estimations. Some sample estimation techniques are discussed in more detail in section IV.C.

When using the semijoin operator in the execution tree, it is frequently necessary to join MAP tables so as to ensure that all the required attributes are part of the final result cluster. The join is preformed in general on common virtual addresses. The GSQL command for the join operation involving MAP tables is as follows:

```
CREATE ACTUAL CLUSTER <N1.CN3(k) ANAME3>
SELECT  <SCHEMA1> AS <TABLE1>
FROM    <N1.CN1(i) ANAME1>
        <N1.CN2(j) ANAME2>
WHERE   <BOOLEAN CONDITIONS>                    (26)
```

In this case, N1.CN1(i) and N2.CN2(j) represent the MAP tables which need to be joined and the result MAP table is N1.CN3(k).

In general, one can define the join operation on two clusters. The DAVID DBMS runs through two cluster instances N1.CN1(i) and N1.CN2(j) and selects those cluster rows of N1.CN1(i) and N1.CN2(j) which satisfy specific boolean conditions. The result of this operation is a cluster instance N1.CN3(k) whose schema is defined by SELECT statement of the GSQL query, i.e., the DBMS must choose those tables and attributes specified by the SELECT statement and insert the appropriate attribute values from the cluster rows of both clusters N1.CN1(i) and N2.CN2(j) which matched. The GSQL query representing this primitive is given as follows:

```
CREATE ACTUAL CLUSTER <N1.CN3(k) ANAME3>
SELECT <SCHEMA1>
FROM   <N1.CN1(i) ANAME1>
       <N1.CN2(j) ANAME2>
WHERE  <BOOLEAN CONDITIONS>                    (27)
```

From the current implementation experience, it should be noted that this operation is expensive especially if one considers large clusters.

When building on execution tree, the analyst has two options. The first option is to use the semijoin operation and to subsequently perform the appropriate joins on the MAP tables so as to ensure that all the attributes necessary to answer the

query are in the final result cluster. This final result cluster is attained through the use of the ESTABLISH primitive which is described subsequently. The second option is to find a sequence of join operations. In this case, the analyst does not need to join any MAP tables nor use the ESTABLISH operation to attain the final result cluster. The relative importance of each option needs to be investigated for different types of queries and distributed databases.

If the first option is chosen, the DBMS must create the final result cluster. Once a MAP table is created which serves as a link for all the attributes necessary to answer the query and their respective clusters, the ESTABLISH operation is used to bring together all the relevant data in a final result cluster. The ESTABLISH command uses the MAP table (MAPT) and the clusters associated with that MAP table to define the final result cluster N1.CNR(k). The schema of the result cluster is defined by the ESTABLISH statement of the GSQL command. For the current implementation of the ESTABLISH operation, the SCHEMA of the result cluster is based on the schema of the result cluster requested by the user. The GSQL command for the ESTABLISH operation is given as follows:

```
CREATE ACTUAL CLUSTER  <N1.CNR(k) ANAME>
ESTABLISH  <SCHEMA1>
WITH       <N1.MAP(t) MAPT>
USING      <N1.CN1(i) ANAME1>
           <N1.CN2(j) ANAME2>
            :
           <N1.CNq(n) ANAMEq>                    (28)
```

Finally, the operation which transmits data across the distributed network is the TRANSFER operation. The GSQL command is given as:

```
TRANSFER <Nk.CN1(i) ANAME1>
TO <Nm.CN1(i) ANAME2>                            (29)
```

For each cluster transfered across the network there is a new cluster created at the node to which the data is transmitted. The cluster definitions Nk.CN1(i) and Nm.CN1(i) are identical.

B. Execution Strategy

Starting with the query graph in Figure 3 and using the primitive operations described in the previous section, one can build an execution strategy which answers the query. This means finding the sequence of primitive operations and their respective processing sites in the distributed network so as to answer the query. Due to the fact that for each query one can find many execution strategies it is important to select a performance criterion which the DBMS will attempt to optimize or satisfy. In order to illustrate the generation of an execution tree, let us assume that the DBMS is interested in reducing the amount of data transmitted across the distributed network and that the semijoin primitive will be used as a primitive operation as part of the execution strategy. There are a number of heuristic based rules which can be applied when generating an execution tree. The following rules are in part formulated from the experience gained with the relational case.

a) It is important to move the projection and selection operations as far up the execution tree as possible. This reduces the amount of data which is subsequently manipulated by the database. This heuristic based rule has been documented by a number of researchers (Ullman [31], Ceri and Pelegatti [7], Egyhazy and Triantis [15]).

b) Combine whenever possible, the adjacent selections and

projections over the same cluster instance. This enables the DBMS to fetch each cluster row once and perform both operations simultaneously, reducing the local processing costs.

c) Isolate the maximum common select operations over the same cluster instance. What is meant by a common select operation is the selection on identical attribute fields and identical attribute values for multiple occurrences of the same cluster instance. This enables the DBMS to fetch each cluster row once and perform the common select operation first and subsequently perform any additional projection or selection operations as specified by the query. This rule enables the DBMS to reduce its local processing costs. The implementation of this rule is discussed by Welch [32].

d) Sequence semijoins so as to reduce the amount of data transfer in the distributed network by means of heuristic based rules or math programming algorithms. The effectiveness of either one depends, for the most part, on the capability of estimating the result temporary cluster sizes after each primitive operation. This is discussed in more detail in section IV.C. Examples of math programming formulations for distributed homogenous relational DBMS for the sequencing of joins are presented by Gavish and Segev [16], Ceri and Gottlob [6] and Triantis and Egyhazy [30].

e) The sequence of joins for the result MAP tables should be such that the DBMS can access the data requested by the user through the final MAP table. It is assumed that the sizes of the MAP tables are relatively small as compared to the result cluster sizes.

The result clusters created during the execution process are stored as temporary (TEMP) clusters. The concept of a temporary cluster is that of a group of data tables corresponding to the tables found in a cluster row and a "glue" table which describes the schema of the cluster or more specifically how each table row is associated with the remaining table rows of each cluster row. For example, Figure 5 portrays a temporary cluster instance of a cluster instance for the hierarchical data RCIS of Figure 4. One notices that for each data table row, there is a unique identifier and that the "glue" table is comprised of these unique identifiers indicating the relationships among data table rows. The temporary cluster representation facilitates the data manipulations as far as the primitive operations are concerned. Operations on clusters can be translated to operations on data tables and the respective "glue" tables. It is important, however, to define the general conditions under which clusters can be represented in the temporary cluster form. It is not evident that clusters representing special types of network databases, for example, recursive network databases can necessarily be depicted by the temporary cluster form[4].

REGISTRAR									
COURSE									
					INSTRUCTOR		STUDENT		
SEMESTER	YEAR	DEPT	COURSE#	SEC#	#	INSTRUCT	STUDID	STUDNAME	GRADE
FALL	1979	COURSE (COURSE-1)							
					INSTRUCT (INSTRUCTOR-1)		STUDENT (STUDENT-1)		
		ART	101	001	2124	CARTER L	2143	SMITH Q	A
							7143	JONES H	B
							8143	BROWN T.	A
					INSTRUCTOR (INSTRUCTOR-2)		STUDENT (STUDENT-2)		
		ENG	103	001	3024	GARNER B	2143	SMITH Q	A
							7143	JONES H	C
							8143	BROWN T	B
							4123	TAYLOR R	B
					INSTRUCTOR (INSTRUCTOR-3)		STUDENT (STUDENT-3)		
		HST	103	001	1023	ALLAN R	2143	SMITH Q	B
							7143	JONES H	A
							4123	TAYLOR R	B

Figure 4 [illegible] for RCIS

GLUE TABLE RCIS

## RCIS	#REG	#COURSE	#INSTRUCT	#STUDENT
17	1	3	6	9
18	1	3	6	10
19	1	3	6	11
20	1	4	7	9
21	1	4	7	12
22	1	4	7	13
23	1	4	7	14
24	1	5	8	15
25	1	5	8	16
26	1	5	8	14

REGISTRAR

#REG	SEMESTER	YEAR
1	FALL	1979

COURSE

#COURSE	DEPARTMENT	COURSE#	SEC#
3	ART	101	001
4	ENG	103	001
5	HIST	103	001

INSTRUCTOR

#INSTRUCT	ID#	INSTRUCT
6	2124	CARTER L
7	3024	GARNER B
8	1023	ALLAN R

STUDENT

#STUDENT	STUDID	STUDNAME	GRADE
9	2143	SMITH Q	A
10	7143	JONES H	B
11	8143	BROWN T	A
12	7143	JONES H	C
13	8143	BROWN T	B
14	4123	TAYLOR R	B
15	2143	SMITH Q	B
16	7143	JONES H	A

Figure 5. A Temporary Cluster Instance Representation for RCIS

Figure 6 presents a possible execution strategy for the query represented by the query graph of Figure 3. Additionally, Appendix 3 pictorially depicts the TEMP representation of this execution tree. The first four operations of this execution tree indicate that adjacent selection projections are performed on the four cluster instances N1.REGISTRAR(1),N2.RCIS(1),N3.FAC(1) and N4.STUDINFO(1). These four operations were chosen in adherance to the rules a and b presented earlier. As a result of each adjacent projection selection, the temporary clusters N1_TEMP.REGISTRAR(1), N2.TEMP_RCIS(1), N3.TEMP_FAC(1) and N4_TEMP.STUDINFO(1) are created. These operations result in the query graph of Figure 7, where for each node there exists a result temporary cluster. Notice that we have created the temporary cluster representations N3.TEMP_FAC(1) and N4.TEMP_STUDINFO(1) which represent relational databases from the clusters N3_FAC(1) and N4_STUDINFO(1) which represent network and hierarchical databases respectively. This is done under the assumption that we are achieving the greatest data reduction and that this will facilitate the subsequent semijoin operations.

```
(1)
CREATE ACTUAL CLUSTER N1.TEMP_REGISTRAR(1) R11
SELECT (STUDID) AS REG
FROM N1.REGISTRAR(1) R1
WHERE R1-REG.DEPT = 'MATH'
      R1-REG.GRADE = 'B'

(2)
CREATE ACTUAL CLUSTER N4.TEMP_STUDINFO(1) S1
SELECT (STUDID) AS STUDENT
FROM N4.STUDINFO(1) S

(3)
CREATE ACTUAL CLUSTER N2.TEMP_RCIS(1) RCIS1
SELECT (COURSE) AS REGISTRAR
      (INSTRUCTOR,STUDENT) AS COURSE
      (INSTRUCT) AS INSTRUCTOR
      (STUDID) AS STUDENT
FROM N2.RCIS(1) RCIS
WHERE RCIS-REG-COURSE-STUDENT.GRADE = 'B'
      RCIS-REG-COURSE.DEPT = 'ART'

(4)
CREATE ACTUAL CLUSTER N3.TEMP_FAC(1) F1
SELECT (NAME) AS FACULTY
FROM N3.FAC(1) F
```

```
(5)
TRANSFER N4.TEMP_STUDINFO(1) S1
TO N1.TEMP_STUDINFO(1) S2

(6)
TRANSFER N3.TEMP_FAC(1) F1
TO N2.TEMP_FAC(1) F2

(7)
CREATE ACTUAL CLUSTER N1.TEMP_REGISTRAR(2) R12
                      N1.TEMP_MAP1(1) M1
SELECT * R11
     (# S, # R ) AS MAP1
FROM N1.TEMP_STUDINFO(1) S2
     N1.TEMP_REGISTRAR(1) R11
WHERE S2-FACULTY.STUDID = R11-REG.STUDID

(8)
CREATE ACTUAL CLUSTER N2.TEMP_RCIS(2) RCIS2
                      N2.TEMP_MAP2(1) M2
SELECT * RCIS1
     (#F, #RCIS) AS MAP2
FROM N2.TEMP_FAC(1) F2
     N2.TEMP_RCIS(1) RCIS1
WHERE F2-FACULTY.NAME = RCIS1-REG-COURSE-INSTRUCTOR.INSTRUCT
```

```
(9)
CREATE ACTUAL CLUSTER N2.TEMP_RCIS1(1) RCIS3
SELECT (#RCIS,STUDID) AS REG
FROM N2 TEMP_RCIS(2) RCIS2

(10)
TRANSFER N2.TEMP_RCIS1(1) RCIS3
TO N1.TEMP-RCIS1(1) RCIS4

(11)
CREATE ACTUAL CLUSTER N1.TEMP_RESULT(1) RES
                      N1.TEMP_MAP3(1) M3
SELECT * R12
     (#RCIS, #R) AS MAP3
FROM N1.TEMP_RCIS1(1) RCIS4
     N1.TEMP_REGISTRAR(2) R12
WHERE RCIS4-REG.STUDID = R12-REGISTRAR.STUDID

(12)
CREATE ACTUAL CLUSTER N1.TEMP_FRESULT(1) FRES
ESTABLISH (STUDID) AS RES
WITH MAP3
USING N1.REGISTRAR(1) R1
```

FIGURE 6. Execution Tree for Sample Query

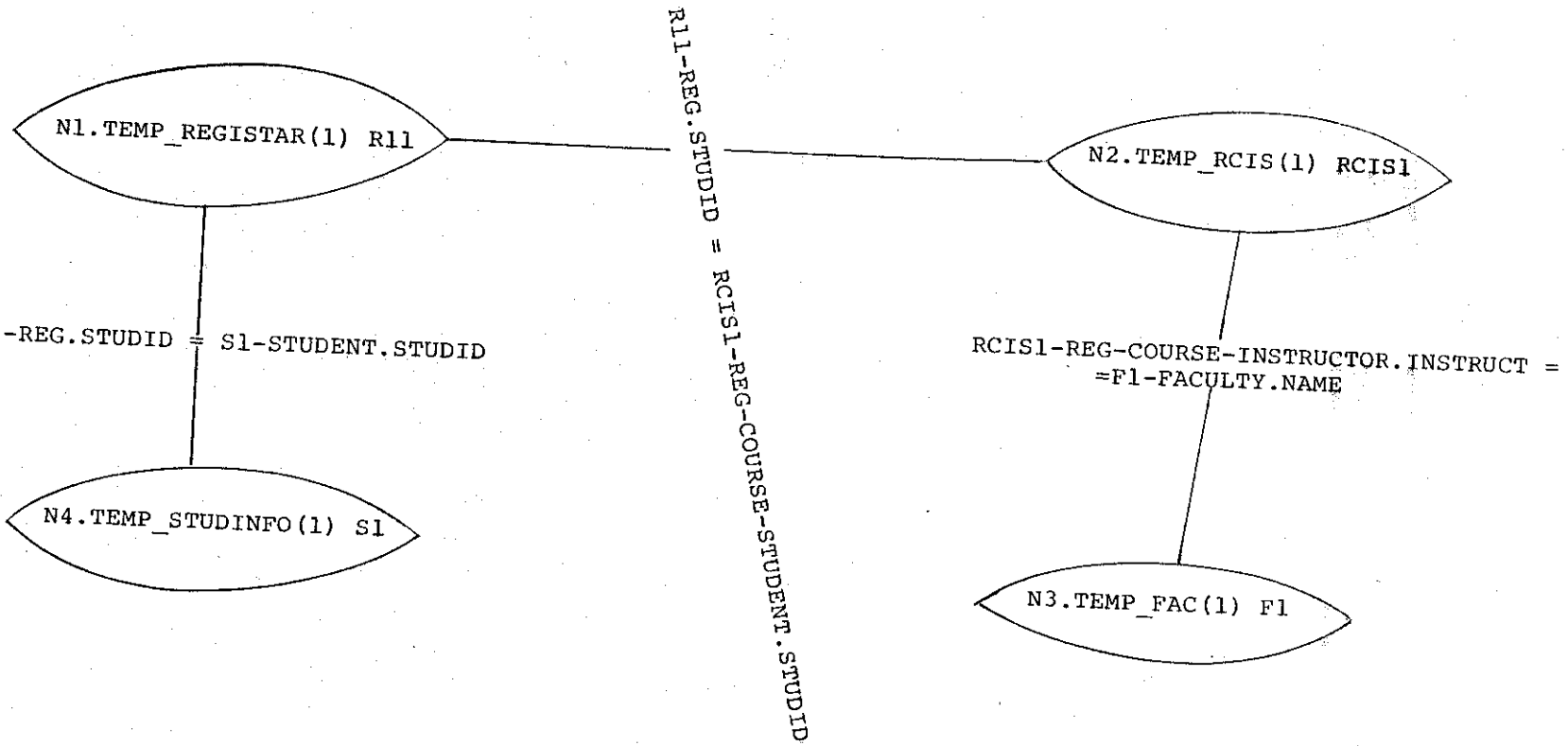

Figure 7. Example Query Graph After Adjacent Selection Projection Operations

After the selections and projections have been performed, the DBMS will select the sequence of semijoins such that the amount of data transfer across the network is minimized or reduced. In our example, N4.TEMP__STUDINFO(1) was transfered to node N1 to be semijoined with N1.TEMP.REGISTRAR(1) whereas, N3.TEMP_FAC(1) was transfered to node N2 to be semijoined with N2.TEMP_RCIS(2). These transfer operations assume that the smallest temporary clusters were transferred across the network. Once N2.TEMP_RCIS(2) was created, a projection operation resulted in the creation of the cluster N2.TEMP_RCIS1(1). This was done in order to achieve additional data reduction. Finally, N2.TEMP_RCIS1(1) was transferred to node N1 to be semijoined with N1.TEMP_REGISTRAR(2) to obtain the final answer to the query. Since both N2.TEMP_RCIS1(1) and N1.TEMP_REGISTRAR(2) have the same size in this example, it does not matter which cluster is transferred across the network.

C. Result Temporary Cluster Size Estimation Issues

The choice of an execution strategy which may reduce for example both the local processing and network transmission costs is contingent upon the capability of estimating the result temporary cluster size after each primitive operation. This assumes that these costs are dependent on the number of bytes transmitted across the network or processed locally by the DBMS. In general, there are two approaches which can be used when estimating temporary cluster sizes. The first approach assumes that the execution strategy will be chosen by the DBMS using statistical information on the original source clusters of the query graph. As the execution strategy is generated, each primitive operation is simulated and the result temporary cluster size is estimated and used as input for the size estimation methodology of the subsequent primitive operations in the execution tree. As one moves further down the execution tree, the accuracy of the result temporary cluster size estimates decreases. This simulated or compiled approach (Jacobs [21]) is used by Egyhazy and Triantis [15] to derive execution strategies for distributed relational DBMSs.

The second approach is to actually execute each primitive operation as the execution tree is being generated. This assumes that the execution strategy is being built in a stepwise fashion using the statistical information of the result temporary

clusters obtained from the execution of the previous primitive operations. In this approach, the accuracy of the result temporary cluster size estimates does not decrease as one moves down the execution tree since the estimation methodology of each primitive operation uses actual cluster size information based on the execution of the previous primitive operations. An example of this approach would be to execute the original selection, projection operations of an execution tree and use the actual sizes of the result temporary clusters to find the sequence of the semijoin operations. This approach can be labelled as a dynamic size estimation methodology and has been discussed briefly by Stoler [28] and Yu and Chang [31] for the relational case. Jacobs [21] refers to this approach as the interpretive approach.

In order to calculate the actual size of a cluster, one needs to know the number of cluster rows and the length of each cluster row. In general, it is not necessary for different cluster rows to have identical lengths. This becomes apparent when one examines the hierarchical and network cases. For example, the length of the cluster row registrar-course-instructor is different from that of the cluster row registrar-course-student of the hierarchical cluster RCIS of Figure 4, since each cluster row is defined by different sets of attributes. Therefore, the result size of a temporary cluster after each primitive operation is determined in general by

finding the expected number of rows of the result cluster as well as the expected length of the result cluster row. The expected number of cluster rows in turn depends on the boolean conditions stated in the WHERE clause of the GSQL primitive, whereas, the expected length of a cluster row depends on the schema of the result cluster defined by the SELECT clause of the GSQL primitive.

The framework proposed by Jacobs [21] is to calculate the following relationship after each primitive operation:

$$TCLSIZE = (CART) * (RF) * (ROW) \qquad (30)$$

where,

TCLSIZE : Expected Temporary Cluster Size

CART : Cardinality of the Source Cluster(s)

RF : Reduction Factor

ROW : Expected Cluster Row Length

The cardinality of the source cluster (CART) times the reduction factor (RF) associated with each primitive operation gives the expected number of rows of the result cluster. The reduction factor for each primitive operation can be expressed in terms of the probability of the boolean conditions being satisfied in the WHERE clause. If the GSQL primitive operation has a number of additive boolean conditions the RF may be expressed as:

$$RF = \prod_{i=1}^{m} PRB_i \qquad (31)$$

Where, m is the total number of boolean conditions in the WHERE clause of GSQL primitive and PRB_i is the probability that the i^{th} boolean condition is satisfied.

The expected result cluster row length (ROW) defined by the SELECT statement of the GSQL primitive query will be the sum of the lengths of each record associated with the result cluster times the probability that the record physically exists in that cluster row, i.e.,

$$ROW = \sum_{j=1}^{n} LR_j * PRL_j \qquad (32)$$

Where, n is the total number of records associated with the result cluster. LR_j is the length of record j and PRL_j is the probability that record j exists in the result cluster row.

Given the above framework, the accuracy of TCLSIZE is dependent on obtaining accurate estimates of PRB_i and PRL_j. This in turn depends on the type of statistics available with respect to the specific records and their occurrences for each source cluster table and for each source cluster row. However, the collection and maintenance of such statistics will increase the

overhead of any DBMS. Therefore, given the original cluster sizes, the types of queries requested by the user and the local processing and network transmission costs, one must assess the necessity of a sophisticated estimation methodology versus a crude one. One approach which is currently under investigation, is to maintain a statistical file or log file which contains the number of occurrences of each attribute value other than key attributes, for each source cluster table and for each source cluster row. Given the current implementation of the TEMP cluster structure the use of a log file is feasible. This approach has been implemented for the relational case by Stoler [28]. However, alternative estimation approaches and their associated methodologies need to be further investigated.

5. SUMMARY AND RESEARCH ISSUES

The query decomposition framework as presented in this paper, is contingent on two major factors. As depicted by Figure 9, the first factor is the data modeling approach which is based on database logic and supports a uniform view of the relational, hierarchical and network models. The second factor is the pre-query decomposition translation which is comprised of three principal translations: complex to basic, external to conceptual and logical optimization. The complex to basic translation takes an original complex query and generates basic queries. Complex queries can be union, nested and tree queries. In the example given in this paper, we translated a nested query into two separate basic queries. The external to conceptual mapping translation generates basic queries which reference conceptual instead of external databases. Finally, the logical optimization translation which was not referenced in our example, is used to replace GSQL queries with smaller but logically equivalent GSQL queries. The strategies involved in accomplishing this translation are based on the database logic representation of a cluster as given in Jacobs [20]. A first implementation of this approach is being contemplated by the DAVID project at NASA's Goddard Space Flight Center.

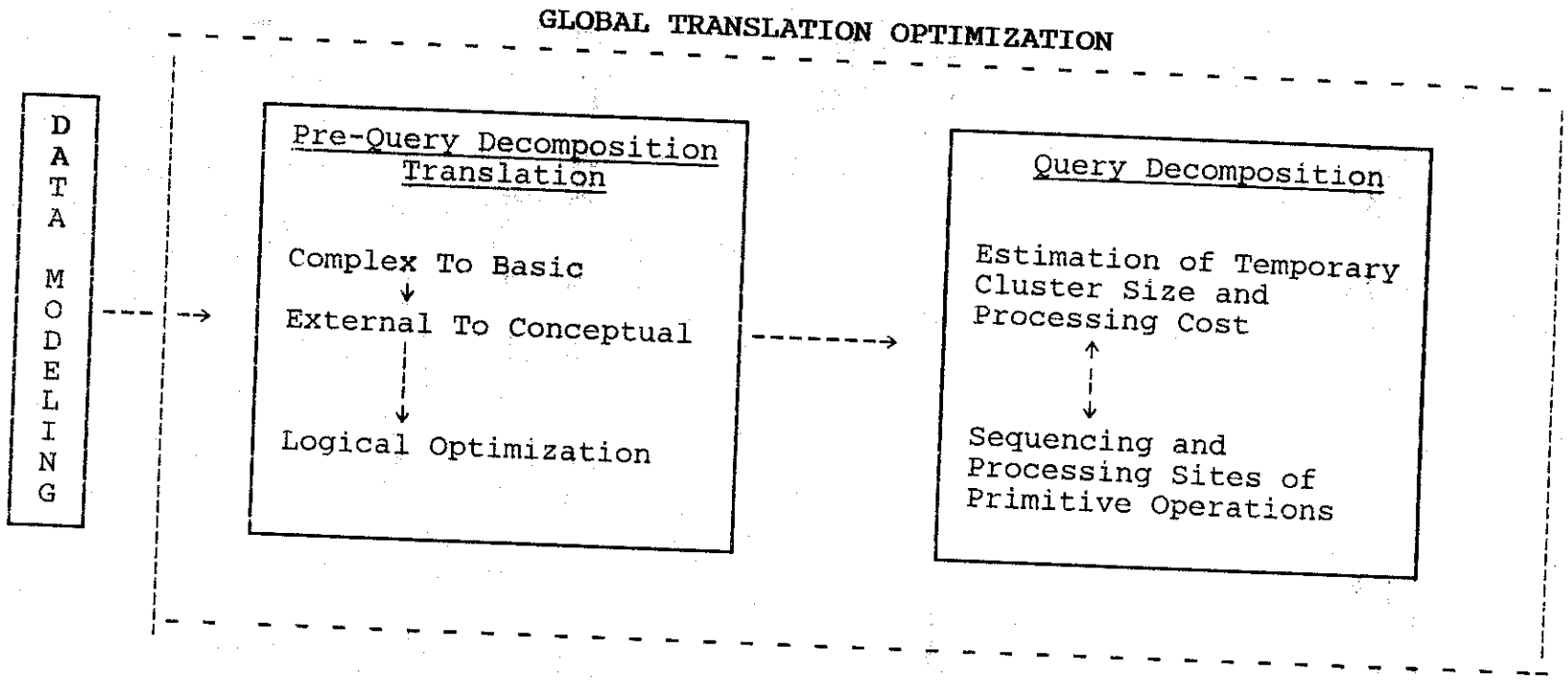

Figure 9. A Framework for Identifying Factors in Query Decomposition Research.

The query generated by the logical optimization translation process becomes the input to query decomposition. As a first approach to query decomposition certain heuristic based rules, as outlined in Section IV.B., may be followed in generating an execution strategy. These rules in conjunction with temporary result size estimation algorithms will determine the query local processing and network transmission costs. These costs are assumed to be directly related to size of a temporary cluster being manipulated. The cluster size and total query processing costs can be subsequently used to model the sequencing of semijoins or joins. The algorithms for estimating the sizes of temporary result clusters as well as determining the sequencing of semijoins or joins are currently under investigation. The output of query decomposition becomes the strategy based on which the DBMS accesses and retrieves the information for the user.

After the initial optimization and size estimation algorithms are implemented and tested for specific queries and database, refinements of these algorithms will be necessary for each of the above described translation processes. However, the major challenge will involve finding methods to improve the global query translation, i.e., from the complex to basic translation to primitive query sequencing. In order to approach this challenge, procedures need to be investigated which will improve the overall translation process and not be as concerned with the optimization of the various translation components.

FOOTNOTES FOR TEXT

[1]Relational databases are databases in which data are represented by one or more separate tables. These tables are called relations and have rows which are called records. In a hierarchical database data are viewed by the user as hierarchies of records. In a hierarchy, a specific record can have subsets of records called children. Each of the children can then have subsets of records as their children. A special property of a hierarchical database is that once a record name is used at some level of the hierarchy, the same record cannot be used anywhere else in the hierarchy. This means that a child record can have only one parent. On the other hand, in a network database a child record can have many parents. Thus, in a network database data appears to the user as sets of records where each record may have many children and itself be the child of many parents.

[2]Relational queries can be partitioned into two classes called tree queries and cyclic queries based on a canonical graph representation of the query [3].

[3]Database logic is used to represent both language and view interpretations. These interpretations can be used to construct mappings between database views.

[4]Recursive network databases are not supported by the current implementation of the DAVID project.

REFERENCES

[1] P. Apers, A. Hevner and B. Yao, "Optimizing Algorithms for Distributed Queries," IEEE Transactions on Software Engineering, vol. SE-11, no. 1., pp. 57-69, 1983.

[2] E. Barkmeyer, M. Mitchell, K. Mikkilinen, S. Su and H. Lam, "An Architecture for Distributed Data Management in Computer Integrated Manufacturing," National Bureau of Standards, Factory Automation Systems Division, NBSIR 86-3312, January, 1986.

[3] P.A. Bernstein and D.N. Chiu, "Using Semi-joins to solve Relational Queries, " Journal of ACM, vol. 28, no. 1, pp. 25-40, 1981.

[4] P.A. Bernstein, N. Goodman, C. Wong and J. Rothnie, "Query Processing in a System for Distributed Databases," ACM Transactions on Database Systems, vol. 6, no. 4, pp. 602-625, 1981.

[5] I. Brodsky, "An Interpretive Implementation of Database Logic: GSQL External to Conceptual Mapping," Masters Thesis, Department of Computer Science, University of Maryland, 1985.

[6] S. Ceri and G. Gottlob, "Optimizing Joins Between Two Partitioned Relations in Distributed Databases," Journal of Parallel and Distributed Computing, vol. 3, pp. 183-205, 1986.

[7] S. Ceri and E. Pelegatti, Distributed Databases: Principles and Systems, McGraw-Hill, New York, 1984.

[8] Tian-Jy Chao and C. J. Egyhazy, "Estimating Temporary File Sizes for Query Graphs in Distributed Relational Database Systems," Proceedings of the International Conference on Data Engineering, Los Angeles, CA, February 5-7, 1986.

[9] C. Chin-Wan and K. Irani, "An Optimization of Queries in Distributed Database Systems." Journal of Parallel and Distributed Computing, vol. 3, pp. 137-157, 1986.

[10] U.W. Chu and P. Hurley, "Optimal Query Processing for Distributed Database Systems," IEEE Transactions on Computers, vol. c-33, no. 9, pp. 835-850, 1982.

[11] C. Codd, "A Relational Model for Large Shared Data Banks," Communications of ACM, vol. 13, no. 6, pp. 377-387, 1970.

[12] C.J. Date, An Introduction To Database Systems, Addison-Wesley, Reading, Mass., 1981.

[13] P. Dwyer, K. Kocsravi and K. Pham, "A Heterogeneous Distributed Database System (DDTS/RAM)," Technical Report CSC-86.7:8216, Honeywell Computer Sciences Center, Golden Valley, MN, 1986.

[14] C. J. Egyhazy, "A Formulation of Query Processing Optimization in Distributed Database Systems," Proceedings of the Fall 1984 Symposium in OR: Traditional and Non-Traditional Applications, Sponsored by NBS, WORMSC, GMU, 1984.

[15] C.J. Egyhazy and K.P. Triantis, "A Query Processing Algorithm for Distributed Relational Data Base Systems," The Computer Journal, 1988, (forthcoming).

[16] B. Gavish and A. Segev, "Query Optimization in Distributed Computer Systems," in Management of Distributed Data Processing, J. Akeka (ed.), North-Holland Publishing Company, Amsterdam, pp. 233-252, 1982.

[17] V. Gliger and G. Luckenburg, "Interconnecting Heterogeneous Database Management Systems," Computer, vol. 12, no. 1, pp. 33-43, 1984.

[18] A.R. Hevner and S.B. Yao, "Query Processing in Distributed Database Systems," IEEE Transactions on Software Engineering, vol. SE-5, no. 3, pp. 177-187, 1979.

[19] ISS, "The Integrated Information Support System." Gateway, vol. 2, no. 2, Industrial Technology Institute, March-April, 1986.

[20] B. Jacobs, Applied Database Logic I: Fundamental Database Issues, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

[21] Ibid, Applied Database Logic II: Heterogeneous Distributed Query Processing, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, (forthcoming).

[22] M. Jarke and J. Koch, "Query Optimization in Database Systems," Computing Surveys, vol. 16, no. 2, pp. 111-152, 1984.

[23] L. Kerschberg, P. Ting and S.B. Yao, "Query Optimization in Star Computer Networks," ACM Transactions On Database Systems, vol. 7, no. 4, pp. 678-711, 1982.

[24] T. Landers and R. Rosenberg, "An Overview of Multibase," in Distributed Databases, H. J. Scheider (ed.), North-Holland Publications, Amsterdam, 1982.

[25] W. Litwin and P. Vigier, "Dynamic Attributes in the Multibase System MRDSM," IEE Computer Society, Proceedings International Conference on Data Engineering, Los Angeles, CA, pp. 103-110, February 5-7, 1986.

[26] National Bureau of Standards, Database Language SQL, ISO/DIS 9075, May, 1986.

[27] J.M. Smith and P.Y. Chang, "Optimizing the Performance of a Relational Algebra Database Interface," Communications of ACM, vol. 18, no. 10, pp. 568-579, 1975.

[28] M. Stoler, "Query Processing Optimization for Distributed Relational Database Systems: An Implementation of a Heuristic Based Algorithm," Master Thesis, Virginia Polytechnic Institute and State University, Department of Systems Engineering, March, 1987.

[29] S. Su, "Modelling Integrated Manufacturing Data With SAM," Computer, pp. 34-49, January, 1986.

[30] K.P. Triantis and C.J. Egyhazy, "An Integer Programming Formulation Embedded in an Algorithm for Query Processing Optimization in Distributed Relational Database Systems," Computers and Operations Research, 1987, (forthcoming).

[31] J. D. Ullman, Principles of Database Systems, 2nd edition, Computer Science Press, Rockville, MD., 1983.

[32] J. N. Welch, "Database Uniformization Problem: Query Decomposition," Masters Thesis, Department of Computer Science, University of Maryland, 1985.

[33] E. Wong, "Retrieving Dispersed Data from SDD-1, "Proceedings of The Second Berkeley Workshop on Distributed Data Management and Computer Networks, Berkeley, California, pp. 217-235, 1977.

[34] C. T. Yu and C.C. Chang, "On the Design of a Query Processing Strategy in a Distributed Database Environment, Proceedings ACM-SIGMOD Conference, San Jose, California, pp. 30-40, June, 1983.

APPENDIX 1

GSQL Cluster Definitions

The remaining clusters of Figure 1 are defined as follows.

```
Define Cluster N1.REGISTRAR(1) R1
REG TABLE
     (SEMESTER    CHAR [ 6];
      YEAR        CHAR [ 4];
      DEPT        CHAR [10];
      COURSE #    CHAR [ 6];
      SEC #       CHAR [ 5];
      ID #        CHAR [10];
      INSTRUCT    CHAR [30];
      STUDID      CHAR [10];
      STUDNAME    CHAR [30];
      GRADE       CHAR [ 5])
     KEY SEMESTER,YEAR,DEPT,COURSE#,SEC#,STUDID
     FD  SEMESTER,YEAR,DEPT,COURSE#,SEC#,STUDID-->GRADE
     STORE AS DBASE (ORACLE);
```

This is also the definition for N5.REGISTRAR(1) R5.

```
Define Cluster N3.FAC(1) F
SYSTEM Table
          (DEPARTMENT TABLE;
           SENIORITY  TABLE;
           FACULTY    TABLE)
           DEPARTMENT TABLE
                      (DEPT    CHAR [10];
                       CHAIR   CHAR [30];
                       OFFICE  CHAR [10];
                       FACULTY TABLE)
                       KEY DEPT
                       FACULTY TABLE
                                (ID#     CHAR [10];
                                 NAME    CHAR [30];
                                 SALARY  NUM  [10];
                                 DEGREE  CHAR [20])
                                KEY ID#
         SENIORITY TABLE
                     (ACADRANK CHAR [10];
                     FACULTY TABLE
                     KEY ACADRANK
                     FACULTY TABLE
                              (ID#     CHAR [10];
                               NAME    CHAR [30];
                               SALARY  NUM [10];
                               DEGREE  CHAR [20])
                              KEY ID#
        FACULTY TABLE
                    (ID#     CHAR [10];
                     NAME    CHAR [30];
                     SALARY  NUM  [10];
                     DEGREE  CHAR [20])
                    KEY ID#
STORE AS DBASE (SEED);
```

```
Define Cluster  N4.STUDINFO(1) S
STUDENT TABLE
          (STUDID          CHAR [10];
           STUDNAME        CHAR [30])
           ADDRESS TABLE;
           BIRTHDATE      CHAR [20];
           DATEADM        CHAR [20];
           CREDITS TABLE;
           GPA            NUM  [10];
           MAJOR          CHAR [10];
           MINOR          CHAR [20])
           KEY STUDID
           ADDRESS TABLE
                          (STREETNO  CHAR [10];
                           STREET    CHAR [20];
                           CITY      CHAR [20];
                           STATE     CHAR [20];
                           ZIPCODE   CHAR [10])
                           KEY  STREETNO, STREET, CITY, STATE
                           FD ZIPCODE --> CITY, STATE
          CREDITS TABLE
                         (DEPT      CHAR [10]
                          NOCREDIT  NUM [ 4])
                         KEY DEPT
STORE AS DBASE (DAVID);
```

APPENDIX 2

Database Logic of The External To Conceptual Mapping Translation for The Example Query

Before we can describe how to arrive at the answer to the query on the external clusters, we must define a conceptual view, V(Dcon), which includes all of the relevant conceptual clusters, an external view V(Dext), which includes the external clusters, and an interpretation of the external view in the conceptual view I(Dext:Dcon).

In the conceptual view, there are several clusters which have tables of the same name. To avoid having to prefix all of the table names with their cluster names, the table names will have a number corresponding to their node appended to the name. For example, the REG table in $N_{\underline{1}}^{1}$.REGISTRAR will be called REG1 and the REG table in RCIS will be called REG2.

In the ensuing discussion we will restrict ourselves to discussing the type and structural definitions and interpretation of views, excluding all issues pertaining to the transformation of character sets (i.e., from EBCDIC to ASCII or vice versa) and the interpretation of constraints.

The conceptual definitions of N1.REGISTRAR(1), N2.RCIS(1),N3.FAC(1) and N4.STUDINFO(1) are given as follows:

V(Dcon) : VIEW DEFINITION

S(Dcon) : SCHEMA DEFINITION

```
TABLE REG1      =  (semester, year, dept, course#, sec#, id#,
                    instruct, studid, studname, grade)
TABLE REG2      =  (semester, year, COURSE2)
TABLE COURSE2   =  (dept, course#, sec#, INSTRUCTOR2,STUDENT2)
TABLE INSTRUCTOR2  =  (id#, instruct)
TABLE STUDENT2  =  (studid, studname, grade)
TABLE SYSTEM3   =  (DEPARTMENT3, SENIORITY3, FACULTY3)
TABLE DEPARTMENT3 =  (dept, chair, office, FACULTY3)
TABLE SENIORITY3  =  (acadrank, FACULTY3)
TABLE FACULTY3    =  (id#, name, salary,degree)
TABLE STUDENT4    =  (studid, studname, ADDRESS4, birthdate,
                      dateadm, CREDITS4, gpa, major, minor)
TABLE ADDRESS4    =  (streetno, street, city, state, zipcode)
TABLE CREDITS4    =  (dept, noncredit)
TABLE REGS        =  (semester, year, dept, course#, sec#, id#,
                      instruct, studid, studname, grade)
```

T(Dcon) : TYPING DEFINITION

```
TYPE SEMTYPE    =  (semester)  ASCII CHAR(6)
TYPE YRTYPE     =  (year) ASCII CHAR(4)
TYPE DEPTYPE    =  (dept, major, minor) ASCII CHAR(10)
TYPE CSTYPE     =  (course#) ASCII CHAR(6)
TYPE SECTYPE    =  (sec#) ASCII CHAR(5)
TYPE IDTYPE     =  (id#, studid) ASCII CHAR(10)
TYPE NAMETYPE   =  (instruct, studname, chair)  ASCII CHAR(30)
TYPE GRDTYPE    =  (grade)     ASCII CHAR(5)
TYPE OFCTYPE    =  (office)    ASCII CHAR(10)
TYPE RANKTYPE   =  (acadrank)  ASCII CHAR(10)
TYPE SALTYPE    =  (salary)    ASCII CHAR(10)
TYPE SNTYPE     =  (streetno)  ASCII CHAR(10)
TYPE STRTYPE    =  (street)    ASCII CHAR(20)
TYPE CITYTYPE   =  (city)      ASCII CHAR(20)
TYPE STATETYPE  =  (state)     ASCII CHAR(20)
TYPE ZIPTYPE    =  (zipcode)   ASCII CHAR(10)
TYPE DATETYPE   =  (birthdate, dateadm) ASCII CHAR(20)
TYPE GPATYPE    =  (gpa) ASCII NUM(10)
TYPE GRDTYPE    =  (noncredit) ASCII NUM(4)
TYPE DEGTYPE    =  (degree)    ASCII CHAR(20)
```

```
P(Dcon):  PREDICATE DEFINITION

   REG1   : (semtype, yrtype, deptype, cstype, sectype, idtype,
             nametype, grdtype) CLUSTER PREDICATE
   REG2-COURSE2-INSTRUCTOR2-STUDENT2
        (semtype, yrtype, COURSE2, deptype, cstype, sectype,
         INSTRUCTOR2, idtype, nametype,STUDENT2, idtype, nametype,
         grdtype) CLUSTER PREDICATE
   SYSTEM3-DEPARTMENT3-SENIORITY3-FACULTY3   :
        (deptype, nametype, ofctype, DEPARTMENT3, ranktype
         FACULTY3, idtype, nametype, saltype, degtype) CLUSTER
         PREDICATE
   SYSTEM3-SENIORITY3-FACULTY3   :
        (ranktype, FACULTY3, idtype, nametype, saltype, degtype)
         SUBCLUSTER PREDICATE
   SYSTEM3-FACULTY3   :  (idtype, nametype, saltype, degtype)
        ROW PREDICATE
   STUDENT4-ADDRESS4   :  (idtype, nametype, ADDRESS4, sntype,
        strtype, citytype, statetype, ziptype, datetype, datetype,
        gpatype,deptype, deptype) CLUSTER PREDICATE
   STUDENT4-CREDITS4   :  (idtype, nametype, datetype, datetype,
        CREDITS4, deptype, crdtype, gpatype, deptype, deptype)
        CLUSTER PREDICATE
   REG5   :  (semtype, yrtype, deptype, cstype, sectype, idtype,
        nametype, idtype, nametype, grdtype) CLUSTER PREDICATE
```

In order to ascertain the existance or absence of an entire cluster, we evaluate cluster predicates. If we are to ascertain the existance or absence of component tables of a cluster, we evaluate subcluster predicates. If we are to ascertain the existance or absence of a row of a cluster, we evaluate row predicates.

The external cluster definitions of N1.REGISTRAR(2) ER1 and N2. RCIS(2) ERCIS are given subsequently. Table REG12 belongs to ER1, whereas, Tables REG22, COURSE22, INSTRUCT22 and STUDENT22 belong to ERCIS.

Similarly, we define V(Dext) as follows:

```
V(Dext)  :  VIEW DEFINITION

S(Dext)  :  SCHEMA DEFINITION

    TABLE REG12  =  (semester, year, dept, course#, sec#, id#,
         instruct, studid, studname, grade)
    TABLE REG22  =  (semester, year, COURSE22, STUDENT22)
    TABLE COURSE22  =  (dept, course#, sec#, INSTRUCTOR22)
    TABLE INSTRUCTOR 22  =  (id#, instruct)
    TABLE STUDENT 22  =  (studid, studname, grade)


T(Dext)  :  TYPING DEFINITION

    TYPE SEMTYPE  =  (semester) ASCII CHAR(6)
    TYPE YRTYPE   =  (year)  ASCII CHAR(4)
    TYPE DEPTYPE  =  (dept)  ASCII CHAR(10)
    TYPE CSTYPE   =  (course#) ASCII CHAR(6)
    TYPE SECTYPE  =  (sec#)  ASCII CHAR(5)
    TYPE IDTYPE   =  (id#, studid)  ASCII CHAR(10)
    TYPE NAMETYPE =  (instruct, studname)  ASCII CHAR(30)
    TYPE GRDTYPE  =  (grade)  ASCII CHAR(5)


P(Dext) : PREDICATE DEFINITION

        REG12  :  (semtype, yrtype, deptype, cstype, sectype,
             idtype, nametype, idtype, nametype, grdtype)
                  CLUSTER PREDICATE
        REG22-COURSE22-INSTRUCTOR22-STUDENT22
             (semtype, yrtype, COURSE22, deptype, cstype, sectype,
             INSTRUCTOR22, idtype, nametype, STUDENT22, idtype)
             CLUSTER PREDICATE
```

The mapping of the external view into the conceptual view, as defined above, refers to specifying the correspondence of types and structures among an interpretation are the coding section and the defining formulas section. For our example we have,

```
I(Dext;Dcon)  :  INTERPRETATION DEFINITION

    EXTERNAL VIEW IS V(Dext)
    CONCEPTUAL VIEW IS V(Dcon)

    CODING SECTION

       CODE FOR  semtype  is  semtype
       CODE FOR  yrtype   is  yrtype
       CODE FOR  deptype  is  deptype
       CODE FOR  cstype   is  cstype
       CODE FOR  sectype  is  sectype
       CODE FOR  idtype   is  idtype
       CODE FOR  nametype is  nametype
       CODE FOR  grdtype  is  grdtype
       CODE FOR  course22 is  deptype, cstype, sectype
       CODE FOR  instructor 22  is  idtype, nametype
       CODE FOR  student 22  is idtype, nametype, grdtype


  DEFINING FORMULAS SECTION


     PREDICATE COURSE22
          ARGUMENTS ARE  :  (semester:1, year:1, dept:1, course#:1,
               sec#:1)
     IS DEFINED BY  :
          (E) id#: 1  (E) instruct:1  (E)  id#:2  (E) salary:1
          REG2-COURSE2-INSTRUCTOR2-STUDENT2
               (semester:1, year:1, dept:1, course#:1, sec#:1,
                id#:1, instruct:1, id:1, name:1, grade:1)
          & SYSTEM3-FACULTY3(id#:2, instruct:1, salary:1)
     PREDICATE INSTRUCTOR22
          ARGUMENTS ARE  :  (semester:1, year:1, dept:1, course#:1,
               sec#:1, id#:1, instruct:1)
          IS DEFINED BY:
               (E)  id#:2  (E) salary:1
               REG2-COURSE2-INSTRUCTOR2
                    (semester:1, year:1, dept:1, course#:1, sec#:1,
                     id#:1, instruct:1)
               & SYSTEMS3-FACULTY3(id#:2, instruct:1, salary:1)


     PREDICATE STUDENT22
          ARGUMENTS ARE :  (semester:1, year:1, dept:1, course#:1,
               section#:1, id#:1, studid:1, studname:1, grade:1)
          IS DEFINED BY  :
               (E) id#:2 (E) salary:1 (E) instruct : 1
               REG2-STUDENT2
                    (semester:1, year:1, dept:1, course#:1,
                     section #:1, id#:1, studid:1, studname:1,
                     grade:1)
               & SYSTEM3-FACULTY3(id#:2, instruct:1, salary:1)
```

```
PREDICATE  :  REG12
     ARGUMENTS ARE  :  (semester:1, year:1, dept:1, course#:1,
                        sec#:1, Id#:1, instruct:1, studid:1, studname:1,
                       grade:1)
     IS DEFINED BY  :
          (E) studname:2
          REG1 (semester:1, year:1, dept:1, course#:1, sec#:1,
          id#:1, instruct:1, studid:1, studname:1, grade:1)
          & STUDENT4(studid:1, studname:2)


PREDICATE  :  REG22-COURSE22-INSTRUCTOR22-STUDENT22
     ARGUMENTS ARE : (semester:1, year:1, dept:1, course#:1
          sec#:1, id#:1, instruct:1, studid:1, studname:1,
          grade:1)
     IS DEFINED BY  :
          (E) id#2  (E) salary:1
          REG2-COURSE2-INSTRUCTOR2-STUDENT2
               (semester:1, year:1, dept:1, course#:1, sec#:1,
                id#:1, instruct:1, studid:1, studname:1,
                grade:1)
          & SYSTEM3-FACULTY3(id#:2, instruct:1, salary:1)
```

The basic idea being that a question about whether or not a row is in a table in the external view reduces to questions about codes in the conceptual view. For example, consider the row predicate of (semtype,yrtype, COURSE22,deptype,cstype,sectype, INSTRUCTOR22,idtype,nametype, STUDENT22,idtype, nametype,grdtype) as:

```
(semester:1,year:1, COURSES22 (dept:1,course#:1,sec#:1)
 INSTRUCTOR22 (id#:1,name:1), STUDENT22(studid:1,studname:1,
                                        grade:1))
```

The defining formula for the question, from above, has arguments:

```
(semester:1,year:1,dept:1,course#:1,sec#:1,id#:1,
 instruct:1,studid:1,studname:1,grade:1)
```

Notice how the variables of V(Dcon), namely semester, year and so on, correspond to the codes for the original variables in V(Dext). The defining formula for this row predicate is:

(E) id#:2 (E) salary:1

```
REG2-COURSE2-INSTRUCTOR2-STUDENT2
(semester:1,year:1,dept:1,course#:1,sec#:1,id#:1,instruct:1,
 studid:1,studname:1,grade:1)
& SYSTEM3-FACULTY3 (id#:2,instruct:1,salary:1)
```

Thus, each table or cluster name is defined in terms of free and existentially quantified variables. Arguments, such as semester:1, corresponds to the free variables used, while the variable names preceeded by (E) correspond to the existentially quantified variables, those not used in defining the table or cluster predicate. This notation and concepts are fully developed by Jacobs [21]. Since the external view has tables with the same structure as conceptual tables, we have appended a 2 to the conceptual table name to arrive at the external table name. For example, REG12 is an external table with the same structure as the conceptual table REG1.

The GSQL query requested by the user at node N1 as expressed by relation 11 is as follows:

```
CREATE ACTUAL CLUSTER N1.INFO(1) INF
SELECT (STUDID, STUDNAME, COURSE) AS STUDENT
       (DEPT, COURSE#, GRADE)  AS COURSES
FROM N1.REGISTRAR(2) ER1
WHERE STUDID IN
     (SELECT STUDID FROM N1.REGISTRAR(2) ER1
                         N2 RCIS(2) ERCIS
      WHERE ER1-REG.DEPT  =  ERCIS-REG-COURSE-STUDENT.STUDID
           AND ER1-REG.DEPT  =  'MATH'
           AND ER1-REG.GRADE =  'B'
           AND ERCIS-REG-COURSE.DEPT = 'ART'
           AND ERCIS-REG-COURSE-STUDENT.GRADE = 'B'
```

To arrive at the answer to this query we must express the view V(Dquery) which corresponds to the query, in terms of the external view on which the query is based, namely V(Dext). We will call this the interpretation of Dquery in terms of Dext, I(Dquery;Dext). Since V(Dext) is in turn derived from relevant conceptual clusters defined in V(Dcon), we must then perform an external to conceptual mapping. The function of this mapping is to replace the external databases in a query with their definitions in terms of the underlying conceptual databases.

The schema of V(Dquery) is:

```
TABLE STUDENT  =  (STUDID, STUDNAME, COURSES)
TABLE COURSES  =  (DEPT, COURSE#, GRADE)
```

In the interpretation I(Dquery;Dext), row predicates and the sequence of variables that range over the code for each data type are:

```
DEFINING FORMULA SECTION

    PREDICATE STUDENT-COURSES
       ARGUMENTS ARE (studid:1, studname:1, dept:1,
          course#:1, grade:1)
       IS DEFINED BY
          (E) semester:1  (E) year:1  (E) sec#:1  (E) id#:1
          (E) instruct:1  (E) course#:2  (E) studname:2
          (E) semester:2  (E) semester:3  (E) year:2
          (E) year:3  (E) course:3  (E) sec#:2  (E) id:2
          (E) instruct:2  (E) studname:3
          REG12(semester:1, year:1, dept:1, course#:1,
              sec#:1, id#:1, instruct:1, studid:1,
              studname:1, grade:1)
           & REG12(semester:1, year:1, 'MATH', course#:2,
              sec#:1, id#:1, instruct:1, studid:1,
              studname:2, 'B')
           & REG22-COURSE22-INSTRUCTOR22
              (semester:2, year:2, 'ART', course#:3,
               sec#:2, id#:2, instruct:2)
           & REG22-STUDENT22
              (semester:3, year:3, studid:1, studname:3, 'B')
```

Basically, each defining formula (PREDICATE-name) represents a question of whether or not a row in the cluster or table defined in Dext reduces to questions about rows in cluster(s) or table(s) defined in Dcon. Let's then perform an external to conceptual mapping of this defining formula in terms of the defining formulas for REG12, REG22-COURSE22-INSTRUCTOR22, and REG22-STUDENT22. Performing the substitutions results in the following formula (the existential variables that appear only once are dropped and a '-' is substituted to mean "don't care" in order to simplify the formula; since the predicates are typed, we can figure out what variables are necessary).

```
PREDICATE STUDENT-COURSES
     ARGUMENTS ARE (studid:1, studname:1, dept:1, course#:1
          grade:1)
     IS DEFINED BY
          REG1(-, -, dept:1, course#:1, -, -, -, studid:1
               studname:1, grade:1)
          & STUDENT4(studid:1,-)

          & REG1(-, -, 'MATH', course#:1, -, -, -, studid:1,
               studname:1, 'B')
          & STUDENT4(studid:1, -)
          & REG2-COURSE2-INSTRUCTOR2(-, -, 'ART', -, -, -,-,)
          & SYSTEM3-FACULTY3(-, -, -)
          & REG2-STUDENT2(-, -, studid:1, -, 'B')
          & SYSTEM3-FACULTY3(-, -, -)
```

Renames, eliminating duplicate conjuncts, and eliminating predicates which use all existentially quantified variables results in the following simplified formula which represents the query as it is defined on the conceptual view.

```
PREDICATE STUDENT-COURSES
     ARGUMENTS ARE (studid:1, studname:1, dept:1, course#:1,
          grade:1)
     IS DEFINED BY
          & REG1( -, -, 'MATH', course#:1, -, -, -,
                studid:1, studname:1, 'B')
          &REG2-COURSE2-INSTRUCTOR2(-, -, 'ART', -, -, -, -)
          & REG2-STUDENT2(-, -, studid:1, -, 'B')
```

At this point, the query graph may be drawn.

APPENDIX 3

An Example of the Temporary Cluster Representation of the Execution Tree Depicted by Figure 6

(1)

GLUE R11

# R	# REG
4	1
5	2
6	3

N1.TEMP_REGISTRAR(1) R11

REG

#REG	STUDID
1	3143
2	2143
3	6143

(2)

GLUE S1

# S	#STUDENT
10	7
11	8
12	9

N4. TEMP_STUDINFO(1) S1

STUDENT

#STUDENT	STUDID
7	5143
8	2143
9	6143

(3)

GLUE RCIS1

#RCIS	#REG	#COURSE	#INSTRUCT	#STUDENT
19	13	14	15	16
20	13	14	15	17
21	13	14	15	18

N2.TEMP_RCIS(1) RCIS1

REGISTRAR

REG
13

COURSE

COURSE
14

INSTRUCTOR

# INSTRUCT	INSTRUCT
15	CARTER

STUDENT

#STUDENT	STUDID
16	2143
17	7143
18	8143

(4)

GLUE F1

# F	#FAC
25	22
26	23
27	24

N3.TEMP_FAC(1) F1

FACULTY

#FAC	NAME
22	CARTER
23	GARNER
24	ALLAN

(5)

GLUE S2

# S	# STUDENT
10	7
11	8
12	9

N1.TEMP_STUDINFO(1) S2

STUDENT

# STUDENT	STUDID
7	5143
8	2143
9	6143

(6)

GLUE F2

# F	# FAC
25	22
26	23
27	24

N3.TEMP_FAC(1) F2

FACULTY

# FAC	NAME
22	CARTER
23	GARNER
24	ALLAN

(7)

GLUE R12

# R	# REG
5	2

N1.TEMP_REGISTRAR(1) R12

REG SAME AS IN (1)

N1.TEMP_MAP1(1) M1

MAP1

# S	#R
11	5

(8)

N2.TEMP_RCIS(2) RCIS2

GLUE RCIS2

# RCIS	#REG	#COURSE	#INSTRUCT	#STUDENT
19	13	14	15	16
20	13	14	15	17
21	14	14	15	18

Same REGISTRAR, COURSE, INSTRUCTOR and STUDENT Tables as in (3)

N2.TEMP_MAP2(1) M2

MAP2

# F	#RCIS
25	19
25	20
25	21

(9)

N2.TEMP_RCIS1(1) RCIS3

GLUE RCIS3

#RCIS3	#RCIS
26	19
27	20
28	21

REG

#RCIS	STUDID
19	2143
20	7143
21	8143

(10)

N1.TEMP_RCIS1(1) RCIS4

GLUE RCIS4

#RCIS3	#RCIS
26	19
27	20
28	21

REG

#RCIS	STUDID
19	2143
20	7143
21	8143

(11)

N1.TEMP_RESULT(1) RES

GLUE RES

REG SAME AS IN (1)

# R	# REG
5	2

N1.TEMP_MAP3 M3

# RCIS	# R
19	5

(12)

N1.TEMP_FRESULT(1) FRES

GLUE FRES

# FRES	#RES
32	31

RES

# RES	STUDID
31	2143