QUERY PROCESSING IN HETEROGENEOUS DISTRIBUTED DATABASE
MANAGEMENT SYSTEMS

by
Bharat Bhasker

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for a degree of
DOCTOR OF PHILOSOPHY

in
Computer Science

APPROVED:

C. J. Egyhazy
E. K. Haddad
K. P. Triantis
H. R. Hartson
F. J. Sicel

June, 1992
Blacksburg, Virginia
QUERY PROCESSING IN HETEROGENEOUS DISTRIBUTED DATABASE MANAGEMENT SYSTEMS

by

Bharat Bhasker

Committee Chairman: Csaba J. Egyhazy

Computer Science

(ABSTRACT)

The goal of this work is to present an advanced query processing algorithm formulated and developed in support of heterogeneous distributed database management systems. Heterogeneous distributed database management systems view the integrated data through an uniform global schema. The query processing algorithm described here produces an inexpensive strategy for a query expressed over the global schema. The research addresses the following aspects of query processing: (1) Formulation of a low level query language to express the fundamental heterogeneous database operations; (2) Translation of the query expressed over the global schema to an equivalent query expressed over a conceptual schema; (3) An estimation methodology to derive the intermediate result sizes of the database operations; (4) A query decomposition algorithm to generate an efficient sequence of the basic database operations to answer the query. This research addressed the first issue by developing an algebraic query language called cluster algebra. The cluster algebra consists of the following operations: (a) Selection, union, intersection and difference, which are extensions of their relational algebraic counterparts to heterogeneous databases; (b) Normal-join and normal-projection which replace their counterparts, join and projection, in the relational algebra; (c) Two new operators embed and unembed to restructure the database schema. The second issue of the query translation was addressed by
development of an algorithm that translates a cluster algebra query expressed over the virtual views to an equivalent cluster algebra query expressed over the conceptual databases. A non-parametric estimation methodology to estimate the result size of a cluster algebra operation was developed to address the third issue described above. Finally, this research developed a query decomposition algorithm, applicable to the relational and non-relational databases, that decomposes a query by computing all profitable semi-join operations, followed by the determination of the best sequence of join operations per processing site. The join optimization is performed by formulating a zero-one integer linear program that uses the non-parametric estimation technique to compute the sizes of intermediate results. The query processing algorithm was implemented in the context of DAVID, a heterogeneous distributed database management system.
ACKNOWLEDGEMENT

I express my sincerest gratitude to my advisor, Dr. Csaba J. Egyhazy for his invaluable guidance and constant encouragement during the course of this research. His tireless help was instrumental in bringing this research to a successful completion.

I am deeply indebted to Dr. Kostantinos P. Triantis for his invaluable suggestions in the development of the zero-one integer linear programming based query processing algorithm. I am grateful for his enthusiastic support and constant encouragement.

My special thanks are to Dr. H. Rex Hartson, and Dr. Emile Haddad for providing very useful suggestions that helped me in improving this document, and John Fuller for his editorial contributions. Also, I thank Jack Huang, John Welch, John Fuller, Steve Hu, Shyam Salona, Judy Hu, Irene Lee and Nagi Wakim for helping me in the development of DAVID, a heterogeneous distributed database management system.

I am extremely grateful to Dr. Barry Jacobs for teaching me the intricate issues involved in the heterogeneous distributed database management systems and database logic. Also, I am thankful for his constant encouragement and financial support during the course of this study.

Finally, I wish to express my sincerest gratitude to my wife, Nandita, who encouraged me for years with an angel's patience.
# Table of Contents

1 Introduction ................................................................. 1  
1.1 Background .............................................................. 4  
1.2 Problem Definition ..................................................... 5  
1.3 Related Research ........................................................ 10  
  1.3.1 Heterogeneous Distributed Database Management Systems ... 15  
  1.3.2 Query Optimization ................................................. 19  
    1.3.2.1 Query Optimization in RDDBMSs ......................... 20  
    1.3.2.1.1 Semi-join Based Approaches .......................... 22  
    1.3.2.1.2 Join Based Approaches ................................. 25  
  1.3.2.2 Intermediate Result Size Estimation ..................... 29  
  1.3.2.3 Query Processing in HDDBMSs .............................. 32  
1.4 Outline of the Work ................................................... 35  

2 The Heterogeneous Data Model ........................................... 36  
  2.1 Introduction .......................................................... 36  
  2.2 Database Logic ....................................................... 36  
  2.3 Virtual and Conceptual Views ....................................... 43  
  2.4 Data Manipulation Languages ....................................... 46  
    2.4.1 Generalized Calculus (G CALC) ............................ 47  
    2.4.2 Generalized Structured Query Language ................. 48  
  2.5 Representation of fragmented data ............................... 50  
    2.5.1 Horizontal Fragmentation .................................... 51  
    2.5.2 Vertical Fragmentation ...................................... 53  
    2.5.3 Hybrid Fragmentation ....................................... 55  
  2.6 Schema Architecture ................................................. 57  
  2.7 Heterogeneous Distributed Database Architecture ............. 66  
  2.8 Conclusion ........................................................... 74  

3 The Cluster Algebra ..................................................... 74  
  3.1 Introduction .......................................................... 74  
  3.2 Operations ............................................................ 74  
    3.2.1 Selection ....................................................... 79  
    3.2.2 Normal Projection ............................................. 81  
    3.2.3 Renaming ........................................................ 82  
    3.2.4 Expand .......................................................... 83  
    3.2.5 Embed ........................................................... 83  
    3.2.6 Unembed ......................................................... 85  
    3.2.7 Normalize ....................................................... 87  
    3.2.8 Union ............................................................ 88  
    3.2.9 Intersection ..................................................... 90  
    3.2.10 Difference ..................................................... 92  
    3.2.11 Cross Product ................................................ 93  
    3.2.12 Normal Natural Join ....................................... 95  
  3.3 Conclusion ........................................................... 96  

4 Query Translation: Virtual to Conceptual Clusters ................. 98  
  4.1 Introduction .......................................................... 98  

v
1 Introduction

The Heterogeneous Distributed Database Management Systems (HDDBMSs) provide uniform access to the data stored in the relational, hierarchical and the network data models [23]. A global data manipulation language is used for expressing the queries over heterogeneous data stored in the component database management systems [23]. The query expressed in the global data manipulation language is answered by decomposing the query into a set of basic database operations such as projection, restriction, join and move operations. If all the component databases of an HDDBMS are relational databases then the sequence of basic database operations can be expressed in the relational algebra. Since the component databases of an HDDBMS may contain data stored in the relational, hierarchical and the network data models, a language that can capture and define the meaning of the operations such as selection on a network database or a join of a relational and a hierarchical database is required.

The HDDBMSs integrate the data stored in the component database management systems to form an integrated global schema [61]. Virtual views are used for resolving the conflicts during the integration of the data stored in the component database management systems. Thus, a query expressed over the global schema is likely to have references to the virtual views. In order to evaluate the query, it has to be translated into a semantically equivalent query expressed over the conceptual databases.
Processing these queries involves both the data transmission and local processing. The transmission efforts can be minimized by reducing the amount of data to be moved among the sites. The objective of a query processing algorithm is to generate an execution strategy, made up of the basic database operations, that minimizes the cost of evaluating the query. The true cost of a strategy can be found only after executing the query. Thus, during the process of selecting an optimal strategy, the cost of evaluation has to be estimated.

The design of a system to process queries expressed over heterogeneous databases requires that the issues raised in the above discussion are addressed. These issues are: (1) Formulation of a low-level query language to facilitate the expression of the basic database operations; (2) The translation of an user query with references to the virtual views to an equivalent query with references to the conceptual databases only; (3) Estimation of the sizes of the intermediate results generated during the decomposition process; and (4) The query decomposition algorithm to generate an optimal sequence of the basic database operations for a given user query.

A number of prototypes have been developed to provide uniform access to the heterogeneous databases. Among them are: Multibase [62], Mermaid [67], IMDAS [63], PRECI [64] and DAVID [8]. Current implementations of PRECI, IMDAS and Mermaid provide uniform access to data stored in the relational database management systems only.

The goal of this research is to address the problem of optimal query processing in heterogeneous distributed database management systems consisting of all three
data models. This research makes an important contribution by addressing the four issues raised above and by development and implementation of a query processing and related estimation algorithms in the context of the DAVID system.

The first contribution of this work is the formulation of a low-level query language, called cluster algebra, for the heterogeneous databases. Cluster algebra extends the semantics and scope of the selection, union, intersection and difference operations to the relational, hierarchical and the network databases. The projection and join operations of the relational algebra, however, can not be extended to the heterogeneous databases due to the structural ambiguities in their results [4,42]. The cluster algebra proposed in this research resolves this issue with the development of two corresponding operations called "normal projection" and "normal join". The cluster algebra also defines embed and unembed operations to perform the schema translation among the relational, hierarchical and the network databases. Finally, a grammar for expressing valid cluster algebra queries has been defined as a part of this work.

This research develops and describes a solution to the problem of translating a cluster algebra query expressed over the virtual views to an equivalent query expressed over conceptual databases.

A non-parametric estimation methodology to estimate the size of the result of a cluster algebra operation is also developed and described in this work. The methodology can estimate the result sizes of the cluster algebra queries expressed over the relational, hierarchical and the network databases.
Finally, since the query optimization problem has been shown to be NP-Hard [30], this research proposes, develops and implements a heuristics based algorithm to decompose a query expressed over the relational, hierarchical and the network databases into a sequence of cluster algebra operations. The heuristics tries to minimize the cost of processing a query. Since the query optimization problem is NP-Hard, the sequence of operations generated by this algorithm will be referred to as a good sequence rather than an optimal sequence in the rest of this work.

1.1 Background

Given the tremendous diversity of computers, operating systems, database management systems and communication alternatives in today's non-transparent information environments, computer users have to learn a plethora of methods to access data. This heterogeneity impedes the sharing of information within and across organizational boundaries.

The heterogeneity problem manifests itself in several forms such as (1) Data models (2) Database management systems (3) Network protocols and (4) Operating systems. This work focuses on the query processing in a system, where heterogeneity is due to the data models and database management systems.

A class of solutions called "uniformization" leads to the design of additional systems on top of the already existing component systems that allow the access of information in a uniform way and leave the underlying heterogeneous systems unaffected. In terms of heterogeneous distributed database management systems
(HDDBMSs) the uniformization problem can be stated as follows. Given a collection of HDDBMSs, build interface facilities that will support a uniform access to the set of heterogeneous databases.

Heterogeneous Distributed Database Management systems offer the users uniformized query languages to access the heterogeneous data. The query processing in these systems involves transmission of the data among sites and local processing due to the distributed nature of the data. Results of queries expressed using the uniformized language are generated by performing a sequence of small steps such as transferring, restricting, projecting and joining the data over the network. The efficiency of processing the query is dependent upon the sequence in which these sub-steps are carried out to answer the query.

The "uniformization" of data base management systems has led to the development of heterogeneous distributed database management systems. In the heterogeneous distributed database management systems (HDDBMS) the user has a uniform interface. It is left to the HDDBMS to figure out the location, structure/model of the data, format conversion, inter-lingual translation, virtual to conceptual translation and other factors involved in answering the query.

1.2 Problem Definition

The heterogeneous distributed database management systems offer access of the information without any knowledge of the host database management systems, the location of the data and its environment. In other words, differences in data models, data languages, data format and data storage are transparent to the user.
The uniformization approach has been used to design several prototypes [8,62,63,64,65,67] that assume that the underlying component database management systems are based on one or more of the three most common data models i.e., the relational, hierarchical and network data models.

In order to develop a Heterogeneous Distributed Database Management System (HDDBMS), the system designer needs to deal with the issue of the inter-lingual translation, i.e., given a query expressed in the language of one database management system how can it be translated in terms of another database management system. If all the DBMSs in the distributed environment are relational, then the problem of interpreting relational languages can be solved by couching the relational databases into the formalism of first order logic and subsequently interpreting the various languages under it. The non-relational database models can not be couched in first order logic. This poses additional problems in the design of the heterogeneous distributed database management systems.

The uniform framework, provided by heterogeneous distributed database management systems, allows users to express the desired query in a corresponding uniform language. The query expressed in the uniform language may reference data sets stored in different data models, formats and computer systems. The heterogeneous distributed database management systems allow users to query all the data as if it were stored in a single site in a homogeneous environment.

In order to answer the query, HDDBMSs decompose the query into a set of sub-queries and data transfer operations, where each sub-query can be performed on a single site. The total cost of processing a query is dependent upon the sub-queries
and data transfer operations generated to answer the original query and the sequence in which they are executed. The query processing in HDDBMSs is influenced by the following factors:

1). The Data Models: Meaning whether the HDDBMS supports the relational, hierarchical and the network data models. The query processing algorithms for the HDDBMSs that support non-relational data models, have to take in consideration some additional operations for the translation of the data from one to another model. Additionally, these algorithms have to estimate result sizes of restriction, projection and other operations on non-relational databases.

2). Centralized versus Distributed DBMS: In centralized databases the I/O time is of prime importance while in distributed databases communications cost is the dominating factor. A query processing algorithm in a distributed environment has to minimize inter-site data movement.

3). Replicated Data: If the HDDBMS supports several copies of the same data set over the network then query processing algorithm must have an additional ability to identify the optimal copy for answering the query.

4). Partitioned Data: The support of partitioned data means a data set may be divided in pieces and distributed over the network. For example, the student database may have several partitions stored at each of the local campus sites. The complete information about campus registration can be retrieved by performing a query on the student database but information about a student at a local campus can be obtained by issuing a query on the partition of the student
database stored at the local campus.

5). Optimization Objective: The objective of optimization influences the design of query optimization algorithms. For some systems, the response time may be of prime importance while in others it is the total cost of answering the query.

6). Type of Network: The type of network on which the HDDBMS runs may also influence the query processing. If the HDDBMS runs on some special network configuration the query decomposition algorithm may take advantage of the network topology to schedule the data transfer primitives to reduce the overhead and the cost of data transfers.

7). Virtual Databases: If the HDDBMS supports virtual databases, then additional steps will be required by the query processing algorithm to optimally translate the query in terms of conceptual data bases.

The issues described above are the guiding factors in the design of a query processing algorithm. The objective of this work is to develop a query processing algorithm for a Heterogeneous Distributed Database Management System (HDDBMS) and to address the problem of obtaining a good sequence of operations for query processing in a heterogeneous distributed database management system. The heterogeneous database management system environment assumed in this work supports databases distributed over a long haul network. The databases presented to users are either actual data sets or virtual views of the actual data. The system may also have partitioned data over distributed sites. The heterogeneous data model in this work implies three common data models: relational, hierarchical and network.
The problem of estimating the sizes of intermediate results is embedded in the problem of the query processing in the heterogeneous distributed environment. This is due to the fact that the query processing algorithm has to obtain a good sequence of the operations without actually performing them.

The problem of optimal query processing in the relational distributed database management systems is NP-hard [30]. The relational distributed databases are a special case of heterogeneous distributed databases. Hence, when all the component databases of a distributed heterogeneous database management system are relational databases, the problem of optimal query processing in a heterogeneous database management system is equivalent to the problem of optimal query processing in a distributed relational database management system. Since, the optimal query processing in distributed relational databases is a known NP-hard problem, this implies that the problem of optimal query processing in distributed heterogeneous databases is also NP-hard. Thus, any realistic algorithm for computing a sequence of basic database operations to solve the query will involve use of heuristics. A sequence of operations generated using a heuristics based query processing algorithm will be referred to as a good sequence rather than optimal sequence in this work.

In this research, an algorithm for generating a good sequence of basic databases operations for the user queries is formulated and described. In order to develop a comprehensive query processing algorithm for the heterogeneous database management systems, the following issues need to be resolved: (1) Formulation of a low-level query language to facilitate the expression of the basic database operations; (2) The translation of an user query with references to the virtual views to an
equivalent query with references to the actual databases only; (3) Estimation of the sizes of the intermediate results generated during the decomposition process; and (4) The query decomposition algorithm to generate a good sequence of the basic database operations for a given user query.

This research addresses the first issue by developing and presenting a low level language based on algebra called cluster algebra. The cluster algebra is a generalization of the relational algebra from the relational to the heterogeneous databases. The second issue is addressed by the design and implementation of a query translation algorithm described in chapter 4. The third issue of estimating the sizes of the intermediate results is addressed by the development of a non-parametric estimation technique for the heterogeneous databases. The estimation methodology proposed and implemented as a part of this research is described in section 5.2. Finally, a query decomposition algorithm for the heterogeneous databases is formulated and described in section 5.3.

1.3 Related Research

In the past, various frameworks for dealing with the heterogeneous data have been proposed [1,4,25,27,28,38,42]. Most of these frameworks are based on Mackinouchi’s [38] proposal to abandon the first normal form condition placed on relational databases. Most of these frameworks address only relational and hierarchical data. Some of the general frameworks that couch non-relational data models are described subsequently.
The format model of Hull and Yap [25] describes a mathematical theory for the study of data representation in databases. The format model consists of the formal objects called "formats" along with a specification of how they can be used as a template for structuring the data. The formats are defined using the basic types and three constructors, namely, composition; classification; and collection. The basic types correspond to attributes in the relational model. The composition constructor corresponds to the cartesian product or to the record structure of PASCAL language. The classification constructor corresponds to Pascal's record invariant structure or a discriminated union. The collection constructor is used for the formation of sets of the objects of a given type. These constructors of the format model can be used recursively. A relation can be defined by the collection of a composition of the basic types. The hierarchical model can be defined in the format model by using the alternate layers of collection and composition operations. Thus, the format model can incorporate the relational and hierarchical databases in an uniform fashion.

Database Logic [27,28] extends the theory of first order logic to encompass relational as well as non-relational databases. Since the relational databases have a strong mathematical foundation in First Order Logic, various relational databases can be interpreted in terms of each other. The relational databases follow a common set of axioms and have uniform languages (e.g. relational algebra and SQL). The network and hierarchical databases do not have such a foundation. Thus, they pose a major obstacle to uniformization. Database logic is an extension of first order logic with the capability to couch the network and the hierarchical database management systems in it. Simply stated, database logic is to any database management system what first order logic is to relational database management systems. The common
languages (SQL, QUEL) used for data manipulation and definition are also generalized to handle heterogeneous database models in an uniform way. Two such languages are Generalized Structured Query Language (GSQL) [27,28] and Generalized Calculus (GCALC) [27,28]. These languages let users express their queries in a uniform fashion. For example, a query involving IMS and Oracle can therefore be expressed in the same fashion as one involving only Oracle.

The Object-oriented Semantic Association Model (OSAM*) [63] models objects in the heterogeneous databases in terms of complex data structures, constraints and rules. In this model, a database is a collection of object classes. An OSAM* class is defined by a name, it's structure, operations and knowledge rules. The structure of a class is defined by the associations of the class to other classes. The various associations are Membership, Aggregation, Interaction, Composition, Cross-product, Generalization and data type constructors such as Set, Ordered Set, Vector, Matrix. The relational, hierarchical and network database models can be represented using the OSAM* modelling constructs. Thus, it can be used as a common data model for uniformly accessing the data stored in these three data models. A relational databases can be modeled using Aggregation association. A hierarchical database can be modeled using the combination of Aggregation, Interaction and Generalization associations. Similarly, a network database can be modeled by including a class in two or more generalization associations with different classes. It also supports an SQL like Global Data Manipulation Language (GDML) for expressing the queries. GDML is a high level non-procedural relational query language.
In a comprehensive survey of prototype systems for managing the heterogeneous data, Sheth and Larson [59] have classified these implementations as either a federated or a non-federated architecture. The term federated architecture was first introduced by Heimbigner and McLeod [60] to imply a system based on an organizational model that uses equal and autonomous databases. In federated systems sharing of the data is controlled and defined by explicit interfaces.

The databases can be classified along two orthogonal dimensions: (1) Conceptual and physical structure; (2) Distribution. Along these dimensions four classes of databases can be identified: (a) Logically centralized and physically centralized, these include conventional centralized database management systems such as Ingres and IMS; (b) Logically centralized and physically distributed, the example of these database management systems are homogeneous distributed database management systems such as Ingres/STAR, Distributed Sybase; (c) Logically distributed and physically centralized; (d) Logically distributed and physically distributed. The last case represents the realm of multidatabase systems. The multidatabase systems can have federated or non-federated architectures.

In the non-federated architecture of the multidatabase management systems, each component database management system is fully autonomous. The control and mode of sharing the data completely rests upon the owner of the appropriate database management system. The access protocols, mode and extent of data sharing between any two database management systems are negotiated on a case by case basis [59]. Thus, in the case of non-federated multidatabase systems, there is no need for a common data model and a common data manipulation language. In general, there
is no global schema in the non-federated architecture. The next-generation multidatabase system of the UniSQL [61] corporation is an example of the non-federated architecture.

On the other hand, the federated architecture integrates the local schemata using a common data model. Apart from the local data manipulation language of an autonomous database management system, a common data language is used for expressing the queries over globally integrated schema of the federation. Depending upon the level of integration, the federated databases management systems can be further classified as loosely and tightly coupled federated database management systems. The Distributed Database Testbed System (DDTS) [23] and Mermaid [65] are examples of the tightly coupled federated database management system. The MRDSM [67] is an example of the loosely coupled federated database management system. A federated database management system in which all the local as well as the global schemata can be defined in same data model is referred to as a homogeneous federated database management system. A federated database management system in which one or more constituent do not conform to the same data model is said to be heterogeneous federated database management system.

One important aspect of the heterogeneous federated database management systems is the use of a common framework for the data stored in different data models. The common framework consists of a generalized data model and a corresponding generalized manipulation language. The advantages of this approach include the ability to create a global and consistent view of all the data by mapping all local database schemata into the global model and subsequent use of a common query
language to access the data stored in different data models. In the federated architecture, the mapping and translations between each DBMS and the generalized model and generalized language are sufficient. The software that manages heterogeneous federated databases in a distributed environment is referred to as heterogeneous distributed database management system (HDDBMS).

1.3.1 Heterogeneous Distributed Database Management Systems

In the development of heterogeneous databases management systems several issues are encountered. Some of the important issues are: global data model, global schema construction, query processing, global update and transaction management.

In particular, this work concentrates on the distributed query processing in a system of heterogeneous databases. Heterogeneity in the current context is defined as a system consisting of the following data models: relational, hierarchical and network. The work is performed in the context of the DAVID prototype.

A number of prototypes have been developed during last ten years to address the heterogeneity problem of the databases. Among them are: Multibase (Computer Corporation of America) [62], DDTS (Honeywell) [23], Mermaid (Unisys) [67], ADDS (Ammoco Production Company) [66], IMDAS (National Institute of Standards and Technology) [63], and DAVID (Goddard Space Flight Center, NASA) [8]. Outside the USA, the MRDSM (INRIA, France) [67], and PRECI (University of Aberdeen (Scotland, U.K.) [64] prototype systems exist. There are many differences among these systems with respect to the application they have been
designed for, the local data models and DBMSs supported, the global model definition and the query processing capabilities. The capabilities of these prototype systems are described subsequently.

Multibase, a heterogeneous database management system, provides retrieval capability on the heterogeneous distributed data. The system uses the functional data model for representing global schemata and Daplex as a common high level language for retrieval of the data. Multibase users view the data through this uniform model and manipulate it using Daplex irrespective of the characteristics of the host DBMS in which the data resides. Thus, the system provides complete transparency to the heterogeneity in terms of the database management systems. It also provides an indirect support for fragmented data. Multibase manages all of its data through two components - the Global Data Manager (GDM) and the Local Database Interface (LDI). The queries expressed in Daplex on the global schema are submitted to the GDM. The GDM decomposes a Daplex query on the global schema into a set of Daplex sub-queries on local databases or auxiliary databases. The output queries generated by the GDM refer to a single site. The system dispatches these local database sub-queries to the proper LDIs. The GDM also generates a strategy to assemble the results of all these sub-queries and load the data in proper user format. The LDIs accept Daplex queries referring to a single site as input. An individual LDI has specific knowledge of a particular site. It analyzes and optimizes an individual single site query. It also executes the query and sends the result back to the site of origin.
The Distributed Database Testbed System (DDTS) uses the E-C-R model [23] and GORDAS [23] as the high level generalized language. The system is based on the relational model and thus can not provide support and transparency for non-relational data. The data definition in the DDTS is based on 5-level schema architecture. It has a single federation schema called global representation schema [59]. A user sees only the external level schema. The semantic description of the entire database is described at the conceptual level of the schema. The DDTS system performs semantic query optimization as the first step of processing.

The Mermaid system developed by the Systems Development Corporation, USA, integrates the data stored in the relational database management systems. It uses a four level schema mapping architecture [65] to integrate the component schemata. The SQL and ARIEL are used for formulating queries over the global schema. The Mermaid query optimization algorithm is one of the most comprehensive algorithm amongst the heterogeneous distributes database management system prototypes [65]. It is based on an extension of the SDD-1 [7] algorithm. The algorithm performs initial local processing, followed by the inter-site semi-joins to reduce the communications cost. Finally, all the reduced relations are assembled at one site to generate the final answer. The algorithm employs the selectivity approach for estimating the intermediate result sizes.

The Ammoco Distributed Data System (ADDS) is a heterogeneous distributed database management system developed by Ammoco, USA. The system can integrate the heterogeneous data stored in hierarchical and relational data models. The system does not maintain a global schema. The user queries are formulated and translated
using the information stored in the local export schemata.

The IMDAS prototype developed by the National Institute of Standards and Technology currently supports only the relational database management systems and arbitrary file systems. The global data model used for the integration purposes is Semantic Association Model [63]. The user queries are formulated in an SQL like language. These queries are translated into an internal format and decomposed in such a way that each of the query can be processed by a basic data server. A basic data server is an interface between the individual DBMS and IMDAS. The result of sub-queries are assembled by the distributed data server. If the data requested in a query is beyond the scope of a distributed data server, the query is transferred to the master data server. The scheduling of operations is based on performing parallel execution.

The MRDSM prototype developed at INRIA, France to support multiple databases uses extended relational model as the global data model. It can integrate the databases that are relational or have a relational interface. A query in MRDSM is decomposed into a set of single database sub-queries and inter database sub-queries. The result of single database sub-queries are loaded into a working database. The inter database sub-queries are processed over the working database.

The PRECI system developed at the University of Aberdeen, U.K., uses the canonical data model [64] for integration of the component schema into a global schema. All the databases supported by the system are relational. The query
processing decomposes a query into a set of sub-queries based upon the availability of the data, operators and the communication costs. The algorithm is similar to the one used by the Multibase system. It tries for a high degree of parallelism in execution.

The DAVID system uses database logic as a generalized data model and generalized calculus (GCALC) [28] or generalized SQL (GSQIL) [28] as high level manipulation languages. The database logic framework can model any arbitrary data structure. Thus, the DAVID system supports uniform access to the relational, hierarchical and the network database management systems as well as arbitrary file systems. In terms of the distributed query processing capabilities of the original DAVID prototype, the query decomposition algorithm used was primitive. No semi-join reductions were performed and there was no statistical estimation procedure used to facilitate the determination of a good query processing strategy. In fact, once all the local processing was performed at each node, the temporary clusters were transferred to a predetermined node in the network and were joined on an ad-hoc basis until the final result was obtained. This resulted in very poor performance of the prototype system.

1.3.2 Query Optimization

In the distributed query processing, parts of the databases at dispersed sites have to be transferred to other sites. The query optimization problem in a distributed environment has shown to be NP-hard [30]. Hence, a practical real time algorithm for computing a sequence of operations to perform the query has to rely upon heuristics. In the past ten years, several heuristics based algorithms have been proposed and developed for the distributed relational query processing. In the
distributed query processing algorithms, it is necessary to estimate the sizes of the intermediate relations in order to determine the cost of inter-site data transfer incurred by a step of the query. The query processing in heterogeneous database management systems involves an added complexity due to the fact that the data may be stored in the relational, hierarchical and network data models. In the following sections, a literature survey of the proposed query processing algorithms for Relational Distributed Database Management Systems (RDBMSs), intermediate result size estimation methodologies, and query processing algorithms for Heterogeneous Distributed Database Management Systems (HDBMSs) is described.

1.3.2.1 Query Optimization in RDBMSs

There has been a significant amount of work done in the area of distributed query processing. Most of this work is focused on query processing in the relational data model [2,3,5,6,7,9,13,14,15,17,19,24,36,45,48,51]. These various algorithms try optimizing queries for the Relational Distributed Database Management Systems (RDBMS). Most of the proposed algorithms optimize queries under different sets of assumptions. The algorithms differ in their formulation due to the factors they consider.

The processing differences between various query optimization algorithms can be classified in following categories:

1) Estimation of the size of intermediate results.

2) Use of Semi-join as a reducer.
3) Copy identification in case of replicated data.

4) Suitability of queries for the semi-join.

5) Techniques of translating cyclic queries.

6) Optimal strategies for simple but restricted types of queries.

7) Use of Join operations to optimize a query.

8) Logical and semantic optimization of the query.

One of the first algorithms proposed by Wong [50] decomposed a query on the relational databases into a sequence of move and process operations. In this algorithm, the initial processing site is selected apriori and the cost of moves is calculated. The algorithm iterates until a better lower cost sequence is found. Numerous refinements of this algorithm [3,15,24,48,49] have been proposed. The improved algorithms avoid the need for initial apriori selection of a site. Also, some of these algorithms use improved size estimation techniques for intermediate results.

The algorithms proposed and developed for the query processing in a system for distributed relational databases can be classified in two categories: (1) Semi-join based approaches; (2) Join based approaches. The important algorithms based on Semi-join and Join based approaches are summarized in the following sections.
1.3.2.1.1 Semi-join Based Approaches

A semi-join between two relations \( R_1 \) and \( R_2 \) over an attribute \( A \), denoted by \( R_1 \Join_A R_2 \), implies \( R_1 \) is the reduced and \( R_2 \) the sending relation. The net effect of a semi-join is equivalent to joining \( R_1 \) and \( R_2 \) and then projecting the result over the schema of \( R_1 \).

The algorithm used in the SDD-1 project [7] is based on the semi-join. This algorithm builds a reducer for the query and then selects an assembly site such that the cost of moving all the reduced relations to this site is minimum. Usually, this is the site with maximum reduced data in it. The algorithm decomposes queries expressed over the distributed relational data bases. The estimation technique for the intermediate result sizes assumes uniform distribution for the attribute values and statistical independence amongst attributes.

A query can either be a cyclic or a tree query. A tree query [6,21] is defined by the fact that the query graph of its quantification is a tree in the graph-theoretic sense. If it is not a tree, then an equivalent quantification can be found for which the query graph forms a tree in the graph-theoretic sense. In other words, a query is a cyclic query if the query graph for its quantification and all equivalent quantifications contains a cycle. The algorithms to recognize a tree query have been proposed by Graham [22] and Yu [52]. A cyclic query is transformed into a set of tree queries for generating an optimal solution strategy. The algorithms to transform a cyclic query to a set of tree queries are summarized as follows:
1. The Relation-Merging Algorithm [21,29] states that the relations in the cycles are merged by joining them in order to remove the cycle.

2. The tree Sub-query approach [51] decomposes a cyclic query into a number of tree sub-queries by arbitrarily selecting a relation in the cycle and then building a tree sub-query for each tuple in the relation. The final result is obtained by computing the union of all tree sub-query results.

3. The attribute addition algorithm [30] builds spanning trees with each relation in the cycle as a root and determines the attributes causing the cycle. These attributes are added into the other relations to remove the cycle.

Several researchers have proposed enhancements to the original SDD-1 algorithm for distributed relational queries. The query processing in the replicated data base environment can be done by adding an initial phase to identify the best copy for use in order to keep the communication cost to a minimum. The problem of choosing the minimum number of sites containing all the referred relations in a query is shown to be NP-hard [51].

Yu et al. [51] identified the redundant data movement in the SDD-1 algorithm. These researchers showed that certain relations which are used only for reducing other relations need not be sent to assembly site. Also, the semi-join on the relation already at the assembly site and not used for the reduction of any other relations need not be performed. All these strategies results in improved savings in terms of the communication cost.
There have been various other algorithms which improved upon the techniques used for size estimation of the temporary relations [11,12,16,20,35,37,39,41,46]. This is discussed in a subsequent section.

In the reduction phase, all the above query processing algorithms are heuristics and do not necessarily generate optimal strategies. Chen and Li [14] describe an algorithm to generate an optimal strategy for simple queries. A simple query is one in which all relations appearing in the qualification have the same attribute and each relation has a single attribute. Apers et al. [3] generalized the optimal algorithm to generate strategies for tree queries. The sequence of operations generated by the algorithm is a union of \( n \) sub-strategies where \( n \) is the number of relations referenced by the query. Let \( A \) be a joining attribute of relation \( R \) of the query. A sub-strategy to reduce each relation \( R \) and send it to the result site is generated using the optimal algorithm for simple queries. The final strategy, the union of \( n \) sub-strategies thus generated, is not necessarily optimal. The semi-join strategy constructed above can sometimes be improved using transformations proposed by Luk and Luk [36]. They present an algorithm to transform a given semi-join strategy (produced using some heuristics) into an equivalent strategy such that each semi-join in the former strategy corresponds to a semi-join in the latter strategy and incurs neither higher cost nor lower benefit.

The semi-join based algorithms largely ignore the cost of local processing. The actual cost involving semi-joins is normally higher than the estimated cost. Semi-joins reduce the total data transfer among the nodes, but increase the total number of
messages. In the case of systems where message passing dominates the environment, the cost of message passing may outweigh the benefits of data reductions. This may become pronounced in the case of medium and small size relations.

1.3.2.1.2 Join Based Approaches

The join based approaches optimize queries using a sequence of join and move operations. The join based algorithms can be classified as enumerative and non-enumerative algorithms.

The enumerative algorithms, as the name suggests, enumerate all possible sequences of operations for a given query. The optimal strategy is then selected by finding the operation sequence with the minimum cost. The complexity to compute all operation sequences and their cost is exponential in nature and thus becomes prohibitive for queries with a large number of referenced relations. For the pre-compiled queries that are executed several times, it may be worthwhile to pay the cost of determining the best strategy. Williams et al. [48] proposed and developed an algorithm for the R* that generates the optimal strategy by enumerating all possible strategies. An algorithm proposed by Epstien and Stonebraker [18] generates optimal sequence of operations by partitioning the referred relations into two complementary sets, G1 and G2, where G1 has at least two and G2 has zero or more relations. The algorithm reduces G1 to the best sequence and then moves one relation, R, at a time from G2, thus producing the best sequence for G1 U {R} until G2 contains no relations. In general, the exhaustive enumerative search algorithm scans through e(n) strategies for a query with natural join of n relations, where:
\[ e(1) = e(2) = 1 \]
\[ \sum_{i=2}^{n} \binom{n}{i} \ast \varphi(n - i + 1) \]

where \( \binom{n}{i} \) denotes the number of different combinations for the group having \( i \) relations.

As the number of relations \( n \) grows, \( e(n) \) grows very rapidly and the processing cost for obtaining the best sequence becomes prohibitive. The \( R^* \) algorithm [48] takes into account the local processing cost as well. This algorithm also has a copy identification phase for the replicated relations. Thus, it first identifies which copies are to be used in the case of the replicated relations. Hence it evaluates many more sequences compared to the Epstien and Stonebraker [18] algorithm. In the case of pre-compiled queries the cost pays off over the repeated execution of the query.

Various non-enumerative heuristics have been proposed [5,19] and developed for distributed relational DBMSs (e.g. Ingres). The Heuristic proposed by Baldissera [5] assumes a tree query as input. The algorithm decomposes the tree query into a set of chain queries. The task of converting the tree into a set of chain queries is accomplished by recursively applying the following algorithm:

1. Let \( R \) be the current node of a tree query.

2. If \( R \) has no descendant then Return.

3. If \( R \) has one descendant put \( R \) in the chain and apply the algorithm on the only descendant of \( R \).
4. If R has more than one descendent then put R in each chain and apply the algorithm recursively for each descendent of R.

Once the tree query has been reduced to a set of chain queries, each chain query is further decomposed by the following algorithm:

1. Designate the site with the maximum data referenced in a query as the assembly site.

2. Iterate step 3 through 6 until the chain is reduced.

3. Let C1 be the cost of joining a leaf with its parent and sending the result over to the assembly site.

4. Let C2 be the cost of sending the leaf and parent separately and then joining at assembly site.

5. Compare C1 and C2 to generate the primitive with the lower cost.

6. Modify the query graph to reflect the effect of the primitive.

Another non-enumerative algorithm due to Epstien [19] considers both data communication and local processing costs. This algorithm also takes into account the cyclic queries by transforming them to chain queries. Triantis et al. [45] use integer linear programming to generate an optimal sequence of join operations. The algorithm generates an integer linear program and a set of constraints to model the query. The solution of this formulation is used as an execution plan.
The SDD-1 [7] algorithm processes fragmented relations by decomposing a query into a set of sub-queries, one for each fragment. The result is assembled by computing the union of all sub-query results. The algorithm proposed by Epstein et al. [19] processes horizontally fragmented relations by replicating one of the two fragmented relations involved in a join operation at each fragment site of the other relation. The final result is the union of all such join results. The fragments are built using certain qualifications [10] unique to each site. The qualification may be considered as a constraint on each fragment. The result of a join between the arbitrarily chosen fragments of two relations may be an empty cluster and can be deduced by analyzing the constraints placed on the two clusters. Unfortunately, none of the above algorithms take advantage of this fact. In some cases, the query may be answered without any data transfer.

A more systematic approach to process a query involves logical optimization of the query. In logical optimization a query is transformed into a semantically equivalent query [10,28,47]. The rules for transformations of relational algebra queries can be found in [26]. For the purpose of performing semantic and logical optimization constraint information and semantics of the data is stored in the database. The idea is to apply this information together with transformation rules and produce an equivalent but reduced query. Jacobs [28] has proposed logical optimization of the query based on transformation rules for non-relational databases.

1.3.2.2 Intermediate Result Size Estimation

The goal of query optimization, the derivation of an efficient high level strategy, is achieved by selecting the best alternative amongst the many generated by the
optimizer. The best alternative is the one that minimizes the evaluation of the query. The evaluation of an alternative is a function of a number of parameters such as secondary storage accesses, the communication cost and the central processing units. The actual cost of the evaluation can be derived only by executing the strategy. Optimizers base the evaluation on the estimates of the above factors. Thus, estimation plays a very crucial role in selecting the evaluation plan. Optimizers estimate the size of intermediate results based on the information, estimated or computed, about the database. The quality of the resulting evaluation plan depends to a great degree on the kind of estimation techniques it employs.

The objective of query optimization is to minimize the cost or response time of the query as a whole. Centralized databases have only local processing cost. Thus, the total cost and response time objectives coincide. Certainly this does not hold true for the distributed environment, rendering the problem of optimization far more complex.

The cost of a query in general is contributed by the Input/Output, communication and CPU [44]. The input-output cost is commonly estimated by the number of the physical page reads and writes. Factors such as buffer size and page replacement policies [37] influence the estimation of the physical page references. Whenever a database page is not in the buffer, a logical reference is translated into a physical reference and the page is brought into the buffer. Most of the researchers differ about the role of CPU cost in determining the total query processing cost. Some researchers totally ignore the CPU cost [31] while others base it on some output factors such as total number of resultant tuples [32]. The communication cost has
been considered the single most dominating factor by all the researchers. It is usually estimated by the number of bytes transmitted plus the additional setup cost for the connection [3,6]. A few systems such as R* also consider the cost of passing the messages [34] to assess the total communication cost. Since, the communication cost depends on the number of bytes transmitted, in order to estimate the communication cost a size estimation of temporary intermediate results is mandatory.

The size estimation of the temporary intermediate results is a crucial task in distributed query optimization. The distributed query cost is dominated by the communication cost [3,6]. The communication cost is proportional to the number of bytes sent or received over the network. The size of an intermediate result is estimated by manipulating the profiles of input relations. Mannino et al. [39] view the profiles of a database as complex objects composed of quantitative descriptors. The common descriptors are measures of central tendencies such as the mean, median and mode and of measures of dispersion such as range, variance and standard deviation. Additional descriptors considered are cardinality, density functions such as normal, uniform, binomial, value intervals and counts. For a relational database the relational profile is made up of a list of attribute profiles.

Query optimization researchers have made simplifying assumptions regarding the profiles of databases. The common assumptions made are as follows:

1: Distribution of the attribute values: The attribute values are assumed to be uniformly distributed within the range of the maximum and minimum values.
2: Independence: Two attributes $X$ and $Y$ in any relation are independent of each other. By independence we mean that the conditional probability of a $X$-value given a $Y$-value is the same as the probability of a $X$-value. Or

$$P(X|Y) = P(X)$$

$$P(X \& Y) = P(X) \cdot P(Y)$$

These two assumptions oversimplify the estimation of the sizes of the intermediate results. Under these assumptions, an extreme minimum or maximum value can cause estimated sizes to be very inaccurate. Christodoulakis [16] has shown that the above assumptions produce worst case estimates for relations with distinct attribute values. He also infers that the optimizers based on these assumptions frequently chose the worst strategy. Montgomery et al. [40] have studied the effect of the skewed data on selection and join estimates and report that the uniformity assumption tends to overestimate by 200-300% for the selection operator, but join results were just the opposite. Other researchers [32,37] have reached similar conclusions.

The effect of the correlation of the attributes on the placement of tuples was studied by Vander Zanden [46]. He concluded that the overestimation increases with the correlation.

The statistical literature abounds with similar problems. Mannino et al. [39] in the survey have reported some common tools for profile manipulation in the relational databases. The uniform distribution of attribute assumption have come under severe criticism. The alternate normal, binomial distributions and non-parametric
techniques have also been studied [16,41]. For smaller relational databases even normal and binomial distributions tend to over/underestimate the sizes. For small and medium relational databases Chao and Egyhazy [11] proposed a technique of maintaining the value counts for the attribute values. They maintain meta-data about the database itself. This technique provides computations to estimate the intermediate result sizes. As the size of databases grows larger, the overhead involved in the maintenance of the meta-data and estimation process increases too, thus rendering this technique unsuitable for large databases.

1.3.2.3 Query Processing in HDDBMSs

The query processing algorithms for the heterogeneous distributed database management prototypes [8,62,63,64,65,66] described earlier have been proposed and developed. A review of the query processing algorithms for the prototypes that perform global query processing to produce a good access strategy follows. The prototypes included in the review are: Multibase, Mermaid, IMDAS and DAVID systems.

Multibase query processing algorithm translates a query expressed over global schema into a query processing strategy. The query processing strategy consists of a set of sub-queries, each of the sub-query references exactly one local schema; a set of transfer operations, to move the results of these sub-queries to the site of the query processor; and a set of operations to assemble the final result. The main objective of the algorithm is to minimize the total cost of evaluation for the query. The cost is measured as the sum of the communications and the local processing costs. The query processing algorithm translates the query into an internal representation called
query graph. The next step of the algorithm isolates those sub-queries that can be processed by a single site. Thus, this step generates a set of single site sub-queries. In the next step, the algorithm generates a set of transfer operations to gather the results of the single site sub-queries. The final step consists of generating a set of sub-queries to assemble the result. The algorithm uses the estimation methodology described in [7] to estimate the sizes of intermediate results.

Mermaid query processing algorithm processes heterogeneous databases with data stored in the relational data model and the file systems. The algorithm supports replicated and fragmented relations. It consists of four stages: (1) Site selection; (2) Local reduction; (3) Global reductions; (4) Assembly. The site selection step of the algorithm chooses a copy of the relevant replicated relations for answering the query. The local reduction step is same as what is commonly referred to as initial local processing. This step generates operation to perform the local selection and projection operations on individual sites. The global reduction step uses the hill climbing approach to determine the best sequence of semi-joins. The algorithm adopts the semi-join selection methodology proposed by Bernstein et al. [7]. It uses the selectivity [3,7] approach for estimating the intermediate result sizes. Since the estimation algorithm is imperfect, the algorithm uses a dynamic approach for optimization purposes. This means that the algorithm selects the best semi-join at each iteration and executes it. The actual tuple count returned by the execution is used for computing the size. The next iteration uses the size information returned by the previous step. In the assembly stage all the reduced relations are transferred to the site of the user query and the final query to assemble the result is executed.
IMDAS system uses a SQL like Global Data Manipulation Language (GDML) for expressing the user queries. The Data Manipulation Language Service (DMLS) module of the IMDAS parses the query and generates an equivalent query tree made up of operations on the internal G-relational representation of the schema. The query tree is then checked against the data dictionary to find out whether all the data referenced in the query is present in the underlying basic data servers. In case the query has references to the data not present in the current basic data servers, the query is routed to the master data server to compute the set of basic data servers over which the query is expressed. The query tree may still contain references to the external view of the database. The tree is modified with help of the mapping information to eliminate all the external view references. After this step, the query tree contains references only to the conceptual schema. The query tree is decomposed based on: (1) The operational capability of the basic data servers; (2) The allocation of the fragmented G-relations. The decomposition process also generates a set of move operations. The intermediate results generated by the operations of the previous step are send to Data Assembler, either to assemble the data or to perform further operations on them. Finally, each of the basic data server query is translated into the appropriate language that the designated DBMS can process.

The DAVID uses Generalized Structured Query Language (GSQL) for expressing the global queries over the heterogeneous databases. The query processing algorithm [47] proposed for the DAVID translates the query into an internal representation called query graph. The algorithm generates initial local processing operations to reduce the data movement for the query. In the initial local processing step, it generates the local selection and projection operations on databases stored
at each individual site. In the next step, it generates operations to perform the semi-join operation followed by transfer operation to move all the data to the user query site. In the final step, it assembles the answer from the transferred data.

1.4 Outline of the Work

In the following chapters a model for the heterogeneous databases and query processing algorithms are discussed. In chapter 2 the cluster data model for heterogeneous databases, an architecture of a HDBMS and a functional architecture for query processing are described. Chapter 3 describes a uniform query language, called cluster algebra, developed as a part of this research. The cluster algebra facilitates the expression of the basic database operations for the heterogeneous databases. A solution to the problem of translating a query expressed over virtual clusters to an equivalent query expressed over conceptual clusters is presented in chapter 4. In chapter 5, a query decomposition algorithm for the heterogeneous distributed database management systems proposed, developed and implemented as a part of this research is described. Other important contribution of this work, a non-parametric statistical estimation techniques for estimating the sizes of the intermediate results of the cluster algebra operations, is also described in chapter 5. In chapter 6, the implementation and its comparison to a purely semi-join based algorithm is discussed. Finally, conclusions and directions for the future research in the area of heterogeneous query processing are presented in chapter 7.
2 The Heterogeneous Data Model

2.1 Introduction

For purposes of this work, the heterogeneous databases consist of data sets that are stored in relational, hierarchical and network data models. The existing relational model framework is not powerful enough for modeling the heterogeneous data. Several researcher have proposed extensions of the relational model to represent the data stored in hierarchical and network data models. The format model of Hull and Yap [25], G-relations model of Su [63], and data base logic of Jacobs [27] are few such extensions. The database logic of Jacobs is a logic system general enough to model any arbitrary data structure. This is probably the most general theoretical framework for the databases. A prototype system called DAVID based on the database logic has also been developed at the Goddard Space Flight Center of NASA. In the following sections the database logic [27,28] is introduced. The capability of database logic to model relational as well as non-relational data, the data manipulation languages and representation of virtual views are described in the following sections. The sections 2.2 through 2.4 summarize the database logic developed by Jacobs [27,28].

2.2 Database Logic

The formal logic system developed by Jacobs [27,28], termed database logic, is the model used for representing the heterogeneous data in this work. Intuitively, database logic is the mathematical foundation for the relational, hierarchical, and network models as the first order logic is to the relational model.
In the database logic, the notion of relational i.e. the first order structure, is
generalized to that of database structure. The database view in database logic
corresponds to a theory in the first order logic. A database (relational, hierarchical,
and network) is regarded as a database structure. The idea of interpretation of
database view in the database logic is analogous to the notion of interpretation in
the first order logic.

Formally, a database view \( (V) \) consists of database schema, database language,
and a set of constraints.

\[ V = <S, L, C> \]

Where:

\( S \): is the database schema that describes the data and its structure in
the database.

\( L \): is the language for making assertions about the database.

\( C \): is the set of constraints, which are assertions in the database language
that every instance of the database must obey.

Following definitions further expand on the idea presented above.

**Definition:**

A database schema is a finite collection of rules of the form

\[ S = \{..., R_j = (R_{j1}, R_{j2},..., R_{jk}),......\} \]
The objects \( R_j \) and \( R_{ji} \)'s are referred to as names. Higher order names appear on the left hand side of the rules. The object names, which appear only on the right hand side of the rules are called zero order names. A name \( R_j \) that appears only on the left hand side of the rules, is referred to as an \textit{external name}, otherwise, it is referred to as an \textit{internal name}. For \( S \) to be a database schema, it must satisfy following conditions:

i. No two rules can have same higher order name on the left hand side.

ii. Zero order names and the higher order names are distinct.

iii. Names on the right hand side of a rule are unique.

The language \( L \) for the database view \( V \), used for making assertion on the database schema \( S \), is defined as follows:

\textbf{Definition:}

A database language \( L \) for a database schema \( S \) consists of

1. Domain types \( D \) of schema \( S \)

2. A set of logical symbols consisting of the following:

   i. For each member \( d_i \) of \( D \) we have symbol \( =_{d_i} \);

   ii. \( \exists, \forall, \cap, \cup, -, \Rightarrow, \Leftrightarrow, (, ) \);

   iii. For each higher order name \( R_0 \) in \( S \) a row predicate \( R_0() \);
iv. For each inner name of $R_j$ of $S$, a countable collection of $R_j$ variables;

3. A set of nonlogical symbols consisting of the following:

i. A $n$-place predicate symbol over the domain $D$;

ii. A $n$-place function symbol over the domain $D$;

iii. For each $d_i$ in $D$ of $S$ there is a set of constant symbols $\langle C_{a_i} \rangle$;

Definition:

Let $S = \{..., R_j = (R_{j1}, R_{j2}, ..., R_{jk}), ...\}$ be a collection of rules. A subcluster access path in $S$ is a list of the form $(R_0, R_1, ..., R_i, R_{i+1}, ..., R_r)$ where for $i = 0, 1, ..., r$, the following conditions hold:

i. $R_0$ is an external name.

ii. $R_i$ is the higher order name.

iii. $R_{i+1}$ appears on the right hand side of rule $R_i$.

This definition of schema by Jacobs [27] includes very broad category of structures. In order to restrict it to cover the heterogeneous database structures, following definitions are presented.

Definition:
Let $S = \{..., R_j = (R_{j1}, R_{j2}, ..., R_{jk})_{\ldots}\}$. be a collection of rules that satisfies the properties of database schema. The schema $S$ is a heterogeneous database schema if it satisfies following additional criteria:

i. Let $R_j$ be a name on the left hand side of a rule in $S$. The name $R_j$ does not appear on the right hand side of the same rule.

ii. $S$ can have only one external name in it's rules.

The special classes of schema are defined by placing the restrictions on the set $S$.

**Definition:**

A relational schema $RS$ is a collection of rules $RS = \{R_j = (R_{j1}, R_{j2}, ..., R_{jk})\}$ where $R_{ji}$ for $i = 0, 1, 2, ..., k$ are all zero order names and $R$ is heterogeneous database schema.

**Definition:**

A hierarchical schema $HS$ is a collection of rules $HS = \{..., R_j = (R_{j1}, R_{j2}, ..., R_{jk})_{\ldots}\}$ where:

i. For all $j$, $R_j = (\ldots, R_j, \ldots) \notin HS$

ii. No higher order name appears on the right hand side of two different rules.

iii. For all valid subcluster access paths $(R_0, R_1, \ldots, R_i, R_{i+1}, \ldots, R_r)$ for the schema a higher order name $R_j$ does not appear more than once.
iv. HS is a heterogeneous database schema.

**Definition:**

A network schema \( N \) is a collection of rules \( N = \{ ..., R_j = (R_{j1}, R_{j2}, ..., R_{jk}), ... \} \) where :

i. For all \( j \) \( R_j = (..., R_j, ...) \in N \)

ii. For all valid subcluster access paths \((R_0, R_1, ..., R_i, R_{i+1}, ..., R_r)\) for the schema a higher order name \( R_j \) does not appear more than once.

iii. \( N \) is a heterogeneous database schema.

**The Cluster**

The database structure corresponding to the relational, hierarchical and network database schema is referred to as cluster. In this research, a cluster is defined as follows:

Let \( S \) be a heterogeneous database schema with rules of the form \( S = \{ ..., R_j = (R_{j1}, R_{j2}, ..., R_{jk}), ... \} \)

The zero order names derive their values from the domains which are set of atomic values. Let \( R_j = (R_{j1}, R_{j2}, ..., R_{jk}) \) be a rule in schema \( S \). Let \( D_j, D_{j1}, D_{j2} \) be the domains of \( R_j, R_{j1}, R_{j2} \) respectively. The domain \( D_j \) is a set of k-tuples generated by the cartesian product of domains \( D_{j1}, D_{j2}, ..., D_{jk} \) (i.e. \( D_j = (D_{j1} \times D_{j2} \times ... \times D_{jk}) \)).
The higher order names such as \( R_j \) are set-valued. A cluster is subset of the cartesian product generated by the domains of the names on the right hand side of the rule for the external name in \( S \).

For example, the schema for a hierarchical cluster is given as follows:

Domain of

\[
\begin{align*}
\text{Sem} & \quad \text{is } \{ \text{Fall, Spring, Summer} \} \\
\text{Yr} & \quad \text{is int} \\
\text{Dept} & \quad \text{is Alpha[4]} \\
\text{CRSE#} & \quad \text{is Alphanum[6]} \\
\text{Id#} & \quad \text{is Num[3]} \\
\text{Instruct} & \quad \text{is Alpha[20]} \\
\text{Studid} & \quad \text{is Num[3]} \\
\text{Studname} & \quad \text{is Alpha[20]} \\
\text{Gr} & \quad \text{is } \{ \text{A, B, C, D, F} \}
\end{align*}
\]

\[
\begin{align*}
\text{RCIS} & \quad \text{= (Sem, Yr, Courses)} \\
\text{Courses} & \quad \text{= (Dept, Crse#, Instructor, Student)} \\
\text{Instructor} & \quad \text{= (Id#, Instruct)} \\
\text{Student} & \quad \text{= (Studid, Studname, Gr)}
\end{align*}
\]

In the above example, the domain of Instructor is a set of tuples generated by the domain(\text{Id#}) \times \text{domain(Instruct)}.

The Figure 2-1 shows an instance of this cluster.
Figure 2-1: An instance of rcis cluster.

2.3 Virtual and Conceptual Views

The database view discussed in the earlier sections is either of (1) Conceptual or (2) Virtual type. A conceptual view has a physical database structure associated with it e.g. Registrar cluster shown in Figure 2-1. The conceptual view is denoted by $V(C) = \langle S(C), L(C), C(C) \rangle$.

A virtual view is defined on top of the conceptual views and/or other virtual views. A virtual view may be build on conceptual databases to satisfy the users in need of different structure/model of the same data. A virtual view is denoted by $V(E) = \langle S(E), L(E), C(E) \rangle$. 
Some of the reasons for the existence of virtual views are: providing a customized view, controlled access, conceptual level data independence, a unified view of the fragmented data, a fragmented view of the centralized data and a relational view of the hierarchical and network databases.

Given a university that maintains their student database as a hierarchical structure and a group of departments that wants to view this database as a relational structure, the problem can be resolved by providing a virtual view of the student database with relational schema. The virtual view is defined by a tool referred to as mapping [27] in the database logic. The mapping definition given by Jacobs [28] is summarized as follows:

**Mapping**

Let \( V(E) = <S(E),L(E),C(E)> \) be a virtual view and \( V(C) = <S(C),L(C),C(C)> \) represent a conceptual database view. The mapping between the \( V(E) \) and \( V(C) \) consists of the following:

(a). An 'Interpretation' of the external view into the conceptual view.

(b). A set of operation simulators that translate insertions, deletions and updates from the virtual level to the conceptual level.

The part (b) of the mapping is concerned about the view update translation, while the part (a) solely deals with the query translation.

**Interpretation**
An interpretation \(I(E,C)\) of the language \(L(E)\) of \(V(E)\) into the language \(L(C)\) of \(V(C)\) comprises of the following three components:

i. Coding

ii. Defining formula

iii. Constant transformation

Coding

Each atomic or higher order data type \(T_i'\) in \(L(E)\) is mapped to a sequence of atomic types \(<T_i>\) of \(L(C)\). This sequence is called code for the type \(T_i'\). This means any \(T_i'\) value in the virtual database is represented by a sequence of \(<T_i>\) values in the conceptual database.

Defining Formula

1. For each \((T_0, T_1, \ldots, T_n)\) row predicate \(R_0()\) of \(L(E)\) corresponding to the \(R_0\)-rule \((R_0, R_1, \ldots, R_m)\), define a formula \(Q<R_0;>, <R_1;>, \ldots, <R_n>\) of \(L(C)\). This formula \(R_0()\), where \(<R_i>\) denotes some sequence of variables which range over the code of \(T_i\), is referred to as defining formula for row predicate.

2. For each \((T_1, T_2, \ldots, T_n, T_0)\) function symbol \(f\) we define a formula \(Q<R_0;>, <R_1;>, \ldots, <R_n;>\) of \(L(C)\). We call this the defining formula for \(f\).
3. For each non-logical \((T_0, T_1, ..., T_n)\) predicate symbol \(p\) of \(L(E)\) we define a formula \(Q < R_0 ; >, < R_1 ; >, \ldots, < R_n ; >\) of \(L(C)\). This we call defining formula for the \(p\).

**Constant Transformation**

For each \(T_i'\) of \(L(E)\) and code \(<T_i;>\), we define a procedure \(\text{encode} (T_i')()\). This procedure accepts constant \(C_{ij}^i\) of type \(T_i'\) and returns a sequence of constants \(<C_{ij}^i;>\). Also we define a procedure \(\text{decode} (T_i')()\) which takes output sequence of procedure \(\text{encode} (T_i')()\) and produces original constant \(C_{ij}^i\) of type \(T_i'\).

The coding component as defined above basically describes how a data type in the virtual view is represented by data types of the conceptual view. Similarly, the defining formula component determines which row of the conceptual view is also in the virtual view. Finally, the constant transformation component describes, how the constants of the virtual view are represented in the conceptual view. Given the cluster data model for uniform representation of the heterogeneous data, the uniform data manipulation languages for the model are presented in the following section.

**2.4 Data Manipulation Languages**

For manipulation of databases in a uniform language, a generalization of the relational calculus called GCALC based on database logic has been defined [28]. Also, the SQL for the relational databases has been generalized to the Generalized SQL (GSQL) by Jacobs [28]. Later, in this work a query language based on algebraic notations, called cluster algebra, will be defined.
2.4.1 Generalized Calculus (GCALC)

In addition to language L defined for each schema, a generalized calculous (GCALC) data manipulation language has also been defined by Jacobs[28]. The GCALC is a generalization of the relational calculus from the relational to the heterogeneous case. The GCALC enables us to study and specify the application programs and queries involving multiple database views.

Let \( V = \langle S, L, C \rangle \) be a database view. Also, let \( M \) be a 'Meta variable' corresponding to the database structure, that represents the current database. The general format to query the database in GCALC is as follows:

\[
\text{GET } W \langle R \rangle : Q\langle R \rangle
\]

Where:

\( W \) is a work area for retrieving the data.

\( \langle R \rangle \) are variables in the language \( L \) to be loaded into the work area \( W \).

\( Q\langle R \rangle \) is a Well Formed Formula [27] of \( L \) with non-quantified variables \( \langle R \rangle \) of the language \( L \).

On execution of the above query, the work area \( W \) will contain the tuples of the constant symbols \( \langle e \rangle \) from \( L \) such that

\[
M \models Q \langle e \rangle
\]
The above expression implies that the formula obtained by replacing variables \(<R>\) by constants \(<e>\) is true in the database structure \(M\).

2.4.2 Generalized Structured Query Language

The Generalized SQL for the heterogeneous database management systems is a generalization of the relational SQL. The complete description of the GSQl is given by Jacobs [28]. Here, the query and view definition capabilities of the GSQl are being restated.

The general syntax for querying the data from heterogeneous databases is

```
Select  <Schema_def>
From    <Cluster_list>
Where   <boolean>
```

\(<\text{Schema}_{\text{def}}\> : \ldots, (R_{i1} A_{i1}, R_{i2} A_{i2}, \ldots, R_{ik}) \text{ as } R_i, \ldots \text{ where } R_{ij} \text{ are the internal names from the clusters in the cluster_list or higher order names and the } R_i \text{ 's are the higher order names. The } A_i \text{ 's are the optional alias names, whenever specified the attribute name } R_{ij} \text{ will appear as } A_{ij} \text{ in the result schema}

\(<\text{Cluster_list} \> : C_1, C_2, \ldots \text{ where the } C_i \text{ 's are the cluster names defined in the system.}

\(<\text{boolean} \> : \text{is a well formed formula expressed using the relational and the logical operators on the data elements of the cluster_list.}
Generalized SQL and Interpretations

In our description of virtual views, the process of building virtual views on conceptual databases was described. The process was accomplished by defining an 'Interpretation' between the virtual view and the conceptual clusters. The Generalized SQL can also be used for defining the virtual views on the conceptual clusters. The attributes can be renamed using the aliases in the \(<\text{schema\_def}>\) component of view definition. The syntax for defining virtual views in the GSQL is as follows:

```
Create View <view\_name>
Select <schema\_def>
From <cluster\_list>
Where <boolean>;
```

As stated earlier, the definition of interpretation consists of three parts. The parts are:

1. The conceptual schema \(S(C)\) and conceptual Language \(L(C)\).
2. The virtual schema \(S(E)\) and virtual Language \(L(E)\).
3. The interpretation \(I(E:C)\) of \(L(E)\) in \(L(C)\), consisting of the coding, the defining and the constant transformation section.

Jacobs [28] has shown that all the above three components of the 'interpretation' can be derived from the GSQL query.
The support for the fragmented data can be facilitated by means of defining virtual views. In the following section, various types of fragmentations and their representation using virtual views are described.

2.5 Representation of fragmented data

The two basic types of data fragmentation used in the system are horizontal and vertical fragmentation of the clusters. A hybrid fragmentation can be built by using the combination of these two basic types. The fragments of a cluster can be defined in either the GCALC or the GSQL.

Let rcis-campus[RCIS_CAMPUS] be a cluster where the schema RCIS is defined as follows:

\[
\begin{align*}
\text{Domain of} & \\
\text{Sem} & \text{is} \{\text{Fall, Spring, Summer}\} \\
\text{Yr} & \text{is} \text{int} \\
\text{campus} & \text{is} \{\text{Blacksburg, Telestar, Dahlgren}\} \\
\text{Dept} & \text{is} \text{Alpha}[4] \\
\text{CRSE}\# & \text{is} \text{Alphanum}[6] \\
\text{Id}\# & \text{is} \text{Num}[3] \\
\text{Instruct} & \text{is} \text{Alpha}[20] \\
\text{Studid} & \text{is} \text{Num}[3] \\
\text{Studname} & \text{is} \text{Alpha}[20] \\
\text{Gr} & \text{is} \{\text{A, B, C, D, F}\}
\end{align*}
\]

\[
\begin{align*}
\text{RCIS_CAMPUS} & = (\text{Sem, Yr, campus, Courses}) \\
\text{Courses} & = (\text{Dept, Crse\#, Instructor, Student}) \\
\text{Instructor} & = (\text{Id}\#, \text{Instruct}) \\
\text{Student} & = (\text{Studid, Studname, Gr})
\end{align*}
\]
The fragmentation of an unified cluster is defined as follows:

Let $C$ be the cluster being partitioned, Let $C^p = \{C_1, C_2, C_3, \ldots, C_n\}$ be a set, where $C_i$'s are the fragments of $C^p$. The $C^p$ is a valid partition of $C$ if and only if the following conditions hold:

i. All the data at the uniformized level is mapped into its fragments. There are no data items at the uniformized level that are not covered by its fragments.

ii. It is also possible to rebuild the uniformized level cluster from its fragments.

2.5.1 Horizontal Fragmentation

In the horizontal fragmentation, the cluster rows are partitioned into the non-overlapping subsets. Each subset contains data closely related to a particular site. A valid set of fragments $C^p$ is a horizontal partition of $C$ if it satisfies the following additional conditions.

i. $C = \bigcup_{1 \leq i \leq n} C_i$ where $C_i \in C^p$;

ii. $\forall C_i \in C^p \ S(C_i) = S(C)$

Each fragment cluster $C_i$ is generated by placing a special qualification on the cluster $C$. For example, the rcis_campus[RCIS_CAMPUS] cluster can be partitioned into the following horizontal fragments.
\[ rcis_1 = (rcis; Q_1) \quad Q_1 : \text{campus} = 'Blacksburg' \]
\[ rcis_2 = (rcis; Q_2) \quad Q_2 : \text{campus} = 'Telestar' \]
\[ rcis_3 = (rcis; Q_3) \quad Q_3 : \text{campus} = 'Dahlgren' \]

In the heterogeneous distributed database architecture of Figure 2-2, the unified view of fragments is supported by means of virtual views. The unified virtual view of the above fragments is expressed in GSQL as follows:

Create view RCIS
Select * From RCIS₁
Union Select * From RCIS₂
Union Select * From RCIS₃;

The horizontal fragmentation can also be done indirectly. A qualification \( Q_i \) may utilize conditions from the other clusters. Any such partition is a valid horizontal partition as long as it satisfies all the specified conditions.

2.5.2 Vertical Fragmentation

The vertical fragmentation partitions a unified cluster into the component clusters by grouping its attributes. The group of attributes form a unit cluster. An unified cluster can be partitioned into fragments by projecting the unified cluster over a group of attributes. Also, the unified cluster can be reconstructed from its fragments by joining all the fragments together.
Let $C^P = (C_1, C_2, \ldots, C_n)$ be a partition of $C$. The $C^P$ is a valid vertical partition if and only if the following conditions are satisfied.

Let $A$ be the set of attributes of $C$ and $A_{g_i}$ be the set of attributes of $C_i$ then

1. $A = A_{g_1} \cup A_{g_2} \ldots \cup A_{g_n}$

2. $A_{g_i} \cap A_{g_j} = A_k \quad \forall i \neq j$ Where $A_k$ is set of key attributes.

3. $C_i = \pi_{A_{g_i}}(C) \quad for \quad i = 1, 2, \ldots, n$

4. $C = C_1 \times C_2 \times \ldots \times C_n$

The above definition of vertical partitioning permits replication of only the key attributes amongst the fragment clusters. A less strict definition may permit repetition of some non-key attributes provided there is a justification for it. The less strict definition, if not carefully designed and controlled, may defeat the very purpose of the fragmentation.

In vertical partitioning, the unified cluster can be reconstructed from the fragments as follows:

$$C = C_1 \times C_2 \times \ldots \times C_n[S]$$
Where $S$ is the schema of the unified cluster. This generalized reconstruction mechanism may also be used in reordering the attributes or for changing the presentation view of the database. The idea is further illustrated through the following examples.

Let $\text{Student}_R = (\text{Studid, Campus, Name, Address, Curic_lvl, Dept, Course#, Title, Grade})$ be a relational cluster. A valid vertical partition of the above relation is given as the follows:

- $\text{Studinfo} = (\text{Studid, Campus, Name, Address, Curic_lvl})$
- $\text{Courses} = (\text{Studid, Campus, Dept, Course#, Title, Grade})$

Let $\text{Student}_H = (\text{Studid, Campus, Name, Address, Curic_lvl, Courses ( Dept, Course#, Title, Grade) })$ be a hierarchical cluster. A valid vertical partition of the $\text{Student}_H$ is given as follows:

- $\text{Studinfo} = (\text{Studid, Campus, Name, Address, Curic_lvl})$
- $\text{Courses} = (\text{Studid, Campus, Dept, Course#, Title, Grade})$

Notice the $\text{Student}_R$ and $\text{Student}_H$ may have same valid vertical partition. Thus, if the actual database is being maintained in terms of the fragments Studinfo and Courses then, any or both of the above unified clusters can be defined using GSQL.

Create View $\text{Student}_R$
Select Studid, Campus, Name, Address, Curic_lvl, Dept, Course#, Title, Grade
From Studinfo r, Class s
Where r.Studid = s.Studid;
In the GSQL, the hierarchical view StudentH is defined as follows:

Create View StudentH
Select (Studid, Campus, Name, Address, Curic_i, Courses) as Root, (Dept, course#, Title, Grade) as Courses
From Studinfo r, Class s
Where r.Studid = s.Studid;

2.5.3 Hybrid Fragmentation

The hybrid fragmentation as the name suggest is a combination of the horizontal and vertical fragmentation. A valid hybrid fragmentation must satisfy all the conditions of both the horizontal and vertical fragmentation.

A valid hybrid fragmentation of the StudentR cluster defined above is as follows:

\[ \text{Studinfo}_1 = \text{Studinfo}_{q1} : \text{Campus} = \text{' Blacksburg'} \]

\[ \text{Studinfo}_2 = \text{Studinfo}_{q2} : \text{Campus} = \text{' NOVA'} \]

\[ \text{Courses}_1 = \text{Courses}_{q1} : \text{Campus} = \text{' Blacksburg'} \]

\[ \text{Courses}_2 = \text{Courses}_{q2} : \text{Campus} = \text{' NOVA'} \]

A unified view can be derived from the fragments by the following GSQL definition:
Create View Studinfo_{\text{temp}}
Select * From Studinfo_{1}
Union
Select * From Studinfo_{2};

Create View Courses_{\text{temp}}
Select * From Courses_{1}
union
Select * From Courses_{2};

Create View Student_{R}
Select Studid, Campus, Name, Address, Curic_{lvl}, Dept, Course#, Title, Grade
From Studinfo_{\text{temp}} r, Courses_{\text{temp}} s
Where r.Studid = s.Studid;

In a system of heterogeneous databases the database fragments may be stored in different data models and may have different names for the same attributes. In the following section, the conflicts and discrepancies arising due to defining a global schema for the fragment databases are presented. The identification of the conflicts is prerequisite for developing a schema architecture for the HDDBMS.

2.6 Schema Architecture

The HDDBMSs provide a homogeneous layer on top of the fragment databases, thus giving users the illusion of working on a centralized database management
system. Since the fragment databases are designed independently, the schematic conflicts are mainly due to different representation of the concept, symbolic names for the items and differing need of information.

Assuming a university with three campuses wants to integrate the registrar databases of all the campuses. Although, all registrar databases were designed to keep track of similar information, they were designed, developed and maintained independently. The registrar databases for three campuses are shown in Table 2-1 through 2-3.

Table 2-1: The DAVID Schema Definition of the Registrar cluster at Campus1

```
define cluster Registrar {
  Registrar table {
    Sem char[10];
    Yr char[4];
    Courses table {
      Crse# char[6];
      Cname char[50];
    } Instructor table {
      Id# char[10];
      Instruct char[30];
    };
    Student table {
      Sid char[10];
      Sname char[30];
      Gr char;
    }
  }
} store as david(gbase);
```
Table 2-2: The Oracle Schema Definition of the Registrar Database at Campus2

```
create table Registrar
(
    Sern    char[6];
    Yr      char[4];
    Crse#   char[4];
    Cname   char[50];
    Issn    char[9];
    Instr_name char[30];
    Studid  char[9];
);

create table Studinfo (  
    Studid   Char[9];
    Studname  char[30];
    Gr       char;
);
```
Table 2-3: The Ingres Schema Definition of the Registrar Database at Campus3

create table Registrar (  
  Sem    char[6];  
  Yr     char[4];  
  Course# char[4];  
);  

create table Courses (  
  Course# char[6];  
  Cname   char[50];  
  Issn    char[9];  
  Studid  char[9];  
);  

create table Instruct (  
  Issn    char[9];  
  Instr_name char[30];  
);  

create table Student (  
  Studid  char[9];  
  Studname char[30];  
  Gr      char;  
);
The database stored at campus1 stores the information in a hierarchical data model. Although the information at campus2 and campus3 is stored in the relational model, the two campuses use different number of tables and attribute names. It is clear from the above example, the database users can express the same concept in multiple ways. This makes the task of manipulating data stored in different fragment databases without an unified model and language nearly impossible.

The schema mapping in the Distributed Access View Integrated Database Management System (DAVID) consists of five levels as shown in Figure 2-2. The schema mapping describes the mechanism by which the component heterogeneous database schemata are mapped into the uniform global schema. The level 1 of the architecture consists of heterogeneous database management systems capable of storing different data models such as the relational, hierarchical and the network. The component databases stored at this level are defined in their native data definition languages. For example, the databases described in Table 2-1, 2-2 and 2-3 are stored in DAVID, Oracle and Ingres DBMSs respectively.

The level 2, of the architecture performs mapping of the component database schemata to the common data model. This level maps the native definition of the database components to the uniform schema definition of the database logic presented in the preceding sections. The mapped definition of the component database is referred to as a cluster schema in the database logic. Thus, this layer maps the component databases to the cluster notion of the database logic. The mappings are performed using the data languages of database logic such as GCALC or the Generalized Structured Query Language (GSQL). The level provides an uniform
Figure 2-2: Architecture of a Distributed HDDBMS

layer on top of the distinct database management systems. It also provides transparent view of the data stored in different database management systems. The database
queries can be uniformly expressed over the component databases using the GSQL or GCALC data manipulation languages. For example, the mapped definitions of the Registrar databases stored at three campuses are shown in Table 2-4.

Table 2-4: The Database Logic mapped Registrar Schemas of Three Campuses

Campus1:

    Registrar(Sem,Yr,Course#,Cname, 
            Instructor(Id#,Iname),Student(Sid,Sname,Gr))

Campus2:

    Registrar(Sem,Yr,Course#,Cname,Instr_name,Issn,Studid  
              )
    Studinfo(Studid,Sname,Gr)

Campus3:

    Registrar(Sem,Yr,Course#)  
    Courses(Course#,Cname,Issn,Studid)  
    Instruct(Issn,Iname)  
    Student(Studid,Studname,Gr)

The site mapping layer of the architecture provides the location transparency. The database users use a cluster without specific knowledge about its location. The level 3 of the Figure 2-2 also provides replication transparency. The replication transparency implies that the users are unaware of the replicated copies of the databases. In general, the replication transparency is implied by the site transparency.
This layer maps the three campuses to symbolic names C1, C2 and C3. The qualified names of clusters for the three campuses are C1.Registrar, C2.Registrar and C3.Registrar.

The fourth level of abstraction provides the mapping between the unified schema and the fragment clusters in the system. An unified schema may be made up of several non-overlapping fragment clusters such as the Registrar database shown in the Table 2-4. At this level of the architecture, issues such as name conflicts and the structural conflicts are resolved. If the unified schema is built on pre-existing databases then the type and constraint conflicts are also resolved at this level. The GCALC (generalized Calculus) may be used to define the atomic level mapping between unified schema and its fragments. The GSQL also provides syntax for defining unified schema as well as mapping in a single command. The view definition mechanism described in earlier sections is used for resolving conflicts and defining uniform schemas. For example, assuming the central authorities of the university want to view the campus databases uniformly using the schema of campus1. The "Interpretation Layer" builds virtual clusters for each of the registrar cluster of campus2 and campus3. The virtual clusters have the same schema as the cluster stored at the campus1. The virtual cluster definitions for the campus2 and campus3 databases are shown in Table 2-5.
Table 2-5: Virtual Cluster Definitions for Campus2 and Campus3

```sql
Create view C2.RegistrarView
Select (Sem, Yr, Courses) as Registrar,
       (Crse#, Cname, Instructor, Student) as Courses,
       (Issn Id#, Instr_name Instruct) as Instructor,
       (Studid Sid, Studname Sname, Gr) as Student
From C2.Registrar r, C2.Studinfo s
Where r.Studid = s.Studid;

Create view C3.RegistrarView
Select (Sem, Yr, Courses) as Registrar,
       (Course Crse#, Cname, Instructor, Student) as Courses,
       (Issn Id#, Instr_name Instruct) as Instructor,
       (Studid Sid, Studname Sname, Gr) as Student
From C3.Registrar r, C3.Courses s, C3.Instruct t, C3.Student u
Where r.Course# = s.Course# and s.Issn = t.Issn and s.Studid = u.Studid;
```

The final level of abstraction provides users with a uniform view of all the data accessible to a user of the heterogeneous database management system. At this level, all the database structures stored in different component database management systems are viewed as clusters with the uniform global schema. This level gives users an illusion of a uniform centralized database management system. The user views the data stored in different database management systems, such as Ingres, Oracle, IMS etc., as a uniform structure i.e. cluster. The GCALC and GSQL provide a uniform language for the data manipulation. The GSQL being a high level language is the preferred language for the manipulation of clusters. For example, an unified global schema for all campuses can be built by defining the following virtual cluster.
Create view RegistrarUni
Select *
From C1.Registrar
Union
Select *
From C2.RegistrarView
Union
Select *
From C2.RegistrarView;

In this section, a mechanism to build the global uniform schema on top of heterogeneous distributed databases has been described. In the following section, the system architecture of a heterogeneous distributed database management system called DAVID along with its various modules will be described.

2.7 Heterogeneous Distributed Database Architecture

The system architecture of the DAVID system is shown in Figure 2-3. In the figure a rectangular box represents a DAVID processor and a round ended box represents a query-dependent data structure. The main purpose of the user interface processor is to provide communication with the users. The most common function of the interface is to allow the user to interactively input, edit and execute queries expressed in GSQL. The user can also receive messages from the database (DB) system server concerning the status of a submitted query. Finally, the user interface presents to the user the result of his/her query from DAVID.
The output of the user interface is a command that can have one of the following three forms: a GSQL command statement, an execution tree of GSQL primitives or a program of GSQL commands. The execution tree is a plan of execution ready to be evaluated i.e., parsed and with the necessary cluster definitions included. A program can be in the form of a host programming language or a collection of many GSQL commands.

The database (DB) system server activates other processors to perform certain functions. If the input from the user interface processor is a command program, then the DB system server sends it to the program processor for compilation. If the input is a GSQL command, then it sends the command to the GSQL translation processor for translation. Finally, if the input is an execution tree, it is passed to the execution processor for evaluation. The DB system server can also receive commands from other processors. For example, it can receive execution trees produced by the program processor or the GSQL translation processor as well as DAVID primitives that originate from a foreign node from the DAVID communication processor.

The program processor compiles the input program and produces an execution tree of primitives. In doing so, it activates the GSQL translation processor that translates the GSQL commands. The GSQL translation processor receives GSQL commands that are initially checked for their syntax. Subsequently, the query analysis and decomposition functions are performed. In order to execute these functions, the processor retrieves cluster definitions from the DAVID dictionary.
The DAVID dictionary is stored as a distributed DAVID database cluster. The Dictionary software module provides functions to access, maintain and update the dictionary database cluster.
There are three types of input that can be received by the GSQL translation processor depending on the complexity of the GSQL command. These are primitive, basic and complex queries. Primitive queries involve only one cluster resident in a DBMS. Therefore, these can be evaluated easily. Basic queries (for example, a join of two clusters) can be decomposed into a tree of primitive queries each involving only one cluster and can be evaluated in one pass. Whereas, complex queries (for example, nested queries and queries that involve the UNION operation) are decomposed into one or more basic queries. This is accomplished by the complex-to-basic sub-module. The other important query processing sub-modules are external-to-conceptual and query decomposition.

In a HDDBMS, it is important for the user to be able to view an underlying database according to his/her preferred data model. For example, if an underlying database is hierarchical, the user may want to build an external relational view of the underlying conceptual (actual) database. This means that the user can pretend that his/her 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 database mapping problem can be expressed in terms of database logic [27,28] or GSQL [28]. The external-to-conceptual sub-module translates the external cluster views into conceptual (actual) databases.

The purpose of the query decomposition sub-module is to breakup a query expressed in terms of actual or conceptual databases into its sub-queries.
Additionally, it finds a strategy indicating the sequence of primitive operations and their corresponding processing sites in the network necessary to answer the query. If the query processing strategy is chosen so as to optimize or satisfy some performance criterion such as, response time minimization or reduction then the term query optimization is used. However, since the query optimization problem is in general computationally intractable or NP hard [51] and it relies on statistical information [16,17], most of the algorithms for computing the optimal strategy involve use of heuristics. The heuristic based algorithms produce a good sequence rather than an optimal strategy for solving the query. The issue of query processing in HDDBMSs in general and for the DAVID system in particular is addressed in rest of this work.

Once the query is processed by the GSQL translation processor, an execution tree of primitives is generated that subsequently becomes the input to the execution processor. This processor coordinates the communication and evaluation of primitives. Each primitive query in the tree is examined one at time. The tree is traversed from its leaves to its root. There are two types of primitive queries, i.e., local and remote primitives. They are sent to the corresponding processors for further evaluation and processing. Once the execution tree is evaluated a message is sent back to the DB server.

The local primitive processor may send the primitives it receives to two other processors depending on the type of primitive. The two types of primitives it receives are the local DAVID and the local resident primitives. The local DAVID primitive is a query local to DAVID whereas, the local resident primitive is a query local to a resident DBMS. The DAVID primitive processor executes a DAVID primitive on
DAVID databases. The result can be retrieved by the user interface.

On the other hand, the processing done by the resident primitive processor requires the definition and/or transfer of data from a resident database to DAVID. Depending on the type of primitive (for example, selection, projection, define, insert) and the type of DBMS involved in the query, the resident primitive processor uses a template that includes the syntax of the corresponding primitive operation on the resident database. Once the syntax of the primitive, the actual query and the cluster definitions are obtained, the interface engine is activated. The interface engine loads the template with the appropriate parameter values. The result is a resident "command" file that can be submitted to the DBMS for execution or it might be a "program" that has to be compiled executed in order to load the data into the result buffer. Details concerning the function of the interface engine for relational DBMSs can be found in [8].

The advantages of the template approach can be summarized as follows: (1) It can be used on different DBMSs with one single generic module; (2) It can be used for both host language interfaces and query languages; (3) It can be used for different types of primitive queries; (4) New DBMSs can be easily added to the system by adding the corresponding templates.

The remote command processor accepts two types of primitives. The first, is the "transfer primitive" that requests the transfer of a local DAVID database to another node. The second, is a "remote primitive" which is a primitive on another node of the distributed system. In both cases, a command file is prepared and submitted to the DAVID communication processor.
The communication processor allows the DAVID system to transparently transfer data clusters (containing binary information) and command/messages across heterogeneous computer nodes.

The DAVID system's kernel shown in Figure 2-4 is implemented using a multi-layered approach. It implements the cluster data model and provides operations to create and delete clusters.

The host language interface layer provides operations for the applications programmers of the DAVID system. The operations provided by this interface allow users to perform the following operations from a programming language such as C or PASCAL: open and close clusters; insert and delete cluster rows; update and retrieve selected attributes or all attributes in a cluster.

The Subcluster Access layer implements the cluster data structure using operations provided by the file access layer. This layer provides operations such as, insert, delete, update, and retrieve subcluster, cluster and table rows from a DAVID cluster. The subcluster access module consists of two sets of operations, namely, table row and subcluster row operations. The table row operations are used for the navigation of a cluster. For example, in a hierarchical cluster, the root table is first traversed to establish the currency. Subsequently, all the child table rows associated with the current row can be traversed using the table row operations. On the other hand, the subcluster row operations access all the associated rows for an attribute value in a cluster. A subcluster row consists of data from each table row in its path. The file access layer consists of operations that interact with the host operating system for disk management purposes.
Figure 2-4: DAVID System Software Architecture
Modules shown in Figures 2-3 and 2-4 have been implemented in the current version of the DAVID prototype. The current resident primitive processor allows users to uniformly query the data stored in Ingres, Oracle, IM/DM, Omnibase, Sybase and IMS database management systems.

2.8 Conclusion

The theoretical foundation for couching the relational and the non-relational data models has been presented in this chapter. The schema architecture to represent the heterogeneous distributed data in a uniform way has also been shown in Figure 2-2. A primitive query processing algorithm [47] for the queries expressed in GSQL has been proposed and developed for the DAVID system. In this work, an advanced algorithm for processing the user queries in the system of heterogeneous databases is formulated and developed. Also, a query language based on algebra, namely cluster algebra is proposed to facilitate the development of this advanced query processing algorithm.

The problem of query processing in the distributed heterogeneous environment has been subdivided into three subproblems: (1) Virtual to conceptual cluster translation in the cluster algebra; (2) Estimation of temporary cluster sizes produced by the cluster algebra operations and (3) Query decomposition and optimization. In the following chapters, solutions to each of the three subproblems are described.
3 The Cluster Algebra

3.1 Introduction

Simply stated, cluster algebra is a generalization of the relational algebra. The cluster algebra operations accept clusters as its operands. A cluster algebra operation performs either selection, projection or join of two clusters at a time. Since, the cluster algebra operations perform one function at a time, the cluster algebra provides a greater flexibility in generating optimal sequence for a query. Unlike a GSQL query, where a user may request a combination of all the above in a single query operation.

3.2 Operations

The cluster algebra operations are classified in two categories: the unary and the binary operations. An unary operation of the cluster algebra requires a single operand, i.e. a cluster. Therefore, it can also be viewed as 1-place function. A binary operation of the cluster algebra requires two clusters as operands. The result of all the cluster algebra operations, which is a cluster, is designated as \( r \). As a part of description of these cluster algebra operations a few new terms are also introduced.

\[ \rho_c : \text{denotes a row of the cluster } c \]

\[ \rho_c[R_i] : \text{denotes a value of the attribute } R_i \text{ of the cluster } c \]

In the subsequent sections, the following cluster algebra operations are introduced and defined.
Unary Operations:

1. Selection
2. Normal Projection
3. Renaming
4. Normalize

Binary Operations:

1. Union
2. Difference
3. Intersection
4. Cross Product
5. Normal Natural Join

Some additional operations, used for restructuring the schema, such as Embed, Unembed and Expand are also defined for the cluster algebra. In this work, c[S] is used to denote a cluster c with schema S. Although, it is sufficient to refer a cluster c just by its name, but for the sake of brevity the notation c[S] will be used, where c is name of the cluster and S is the schema of the cluster c. Most of the binary operations require their arguments to be compatible. The concept of compatibility is defined subsequently.

To illustrate the cluster algebra operations in the following sections, rcis_1[RCIS] and rcis_2[RCIS] clusters with the same schema RCIS are defined here. The rules for RCIS schema are as follows:
RCIS = (Sem, Yr, Courses)
Courses = (Dept, Crse#, Instructor, Student)
Instructor = (Id#, Instruct)
Student = (Studid, Studname, Gr)

The Figure 3-1 and Figure 3-2 show instances of the rcis_1 and rcis_2 clusters respectively.

<table>
<thead>
<tr>
<th>Registrar</th>
<th>Courses</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sem</td>
<td>Yr</td>
<td>Dept</td>
</tr>
<tr>
<td>Fall 89</td>
<td></td>
<td>CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
</tr>
<tr>
<td>Spring 90</td>
<td></td>
<td>CS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
</tr>
</tbody>
</table>

Figure 3-1: An instance of rcis_1 cluster

The studinfo[STUDINFO] cluster with relational schema is also used in the examples. The rules of STUDINFO schema are as follows:

STUDINFO = (Studid, Studname, Date_birth, Date_adm)

An instance of studinfo cluster is shown in Figure 3-3.
### Figure 3–2: AN instance of rcis_2 cluster

<table>
<thead>
<tr>
<th>Studid</th>
<th>Studname</th>
<th>Course</th>
<th>Instructor</th>
<th>Gr</th>
<th>Date_birth</th>
<th>Date_adm</th>
</tr>
</thead>
<tbody>
<tr>
<td>042</td>
<td>Hu, Julia</td>
<td>CS 441</td>
<td>Salona, Shyam</td>
<td>A</td>
<td>May 27, 1963</td>
<td>Sep 05, 1988</td>
</tr>
<tr>
<td>048</td>
<td>Hayte, Anne</td>
<td>CS 451</td>
<td>Halem, Milt</td>
<td>B</td>
<td>July 12, 1962</td>
<td>Sep 05, 1987</td>
</tr>
<tr>
<td>052</td>
<td>Smith, P</td>
<td>CS 453</td>
<td>Gordon, Chris</td>
<td>A</td>
<td>Jan 04, 1963</td>
<td>Sep 05, 1988</td>
</tr>
<tr>
<td>099</td>
<td>Ota, Amish</td>
<td>CS 444</td>
<td>Wakam, N</td>
<td>A</td>
<td>June 22, 1964</td>
<td>Sep 05, 1988</td>
</tr>
<tr>
<td>049</td>
<td>Hite, Lee</td>
<td>CS 445</td>
<td>Salona, Shyam</td>
<td>B</td>
<td>Feb 28, 1963</td>
<td>Sep 05, 1988</td>
</tr>
<tr>
<td>052</td>
<td>Smith, P</td>
<td>CS 441</td>
<td>Fuller, J</td>
<td>B</td>
<td>Nov 23, 1964</td>
<td>Sep 05, 1988</td>
</tr>
<tr>
<td>049</td>
<td>Hite, Lee</td>
<td>CS 445</td>
<td>Salona, Shyam</td>
<td>B</td>
<td>Mar 14, 1962</td>
<td>Sep 05, 1988</td>
</tr>
<tr>
<td>110</td>
<td>Goff, Roger</td>
<td>CS 441</td>
<td>Fuller, J</td>
<td>B</td>
<td>Dec 25, 1964</td>
<td>Sep 05, 1988</td>
</tr>
<tr>
<td>150</td>
<td>Wang, Paul</td>
<td>CS 441</td>
<td>Fuller, J</td>
<td>C</td>
<td>July 14, 1966</td>
<td>Sep 05, 1988</td>
</tr>
</tbody>
</table>

### Figure 3–3: An instance of studinfo cluster
Definition: Let $S_1 = (R_1, R_2, R_3, ..., R_k)$ and $S_2 = (R_1, R_2, R_3, ..., R_k, ...)$ be two schemata with higher order names $R_i$ and $R_j$. The higher order names $R_i$ and $R_j$ are said to be equal if the following conditions hold.

Let $P_i$ be the set of all zero order names in the $R_i$ and $Q_i$ be the set of all non-zero order names in $R_i$. Let $P_j$ be the set of all zero order names in the $R_j$ and $Q_j$ be the set of all non-zero order names in $R_j$.

I. $Q_i = \emptyset \iff Q_i = \emptyset$ and $A_k \in P_i \iff A_k \in P_j$ where $A_k$ has $\text{dom}(A_k)$

II. If $Q_i \neq \emptyset$ then

A. $Q_i \neq \emptyset \iff Q_j \neq \emptyset$ and $A_k \in Q_i \iff A_k \in Q_j$ such that $A_k \in Q_i$ and $A_k \in Q_j$ are equal.

B. $A_k \in P_i \iff A_k \in P_j$ where $A_k$ has $\text{dom}(A_k)$.

For example, let RCIS_1 = (Sem, Yr, Courses) and RCIS_2 = (Sem, Courses, Yr) be two database schema where Courses = (Dept, Crse#, Instruct). The higher order name RCIS_1 and RCIS_2 are equal.
Definition: Let $c[S_1]$ and $d[S_2]$ be two clusters with schemata $S_1$ and $S_2$ respectively. The clusters $c[S_1]$ and $d[S_2]$ are considered Compatible if and only if either $S_1$ and $S_2$ are equal or there exists one-to-one and onto mapping $f$ such that:

1. For each $A_i \in S_1$ there exists an $A_j \in S_2$ such that $A_j = f(A_i)$ and $A_i = f^{-1}(A_j)$ where $A_i$ and $A_j$ are the zero-order names and have the same domain.

2. For each $B_i \in S_1$ there exists a $B_j \in S_2$ such that $B_j = f(B_i)$ and $B_i = f^{-1}(B_j)$.

Also, the higher order names $B_i$ and $B_j$ are compatible.

For example, let RCIS_1 and RCIS_2 be two database schema defined as follows:

RCIS_1 = (Sem, Yr, Courses, Student)
Courses = (Dept, Crse#, Credits)
Student = (Studid, Name, Grade);

RCIS_2 = (Sem, Yr, Student, Course1)
Course1 = (Dept, Credit, Course#);

The example schemata defined above are compatible but not equal.

3.2.1 Selection

The selection operation is used to choose some rows from the cluster based on a criteria or condition. The condition is also referred to as a formula. The formal definition of a formula is given as follows:
**Formula Definition:**

Let \( A_i \) be an attribute \( i \) in a database schema \( S \). A *formula* \( F \) in the schema \( S \) is defined as:

1. Let \( K \in \text{Dom}(A_i) \) be a constant, then \( A_i<op>\text{Const} \) is a formula, where \( <op> \in \{= \neq > \geq < \leq \} \).

2. Let \( X \) and \( Y \) be formulas, then the \( X \land Y \), \( X \lor Y \), and \( \lnot X \) are formulas.

3. Nothing else is a formula.

**Definition:** Let \( c[S] \) be a cluster with a schema \( S \) and \( \rho_c \) be a cluster row in \( c \). Then, the *selection* of a cluster \( c[S] \), denoted by \( \sigma_F(c[S]) \), under a formula \( F \) is a subset of \( c[S] \) such that:

\[
\sigma_F(c[S]) = \{ \rho_r \mid \rho_r \in c \text{ and } F \text{ is true for } \rho_r \}
\]

For examples, the result of the operation, \( rcis\_sel = \sigma_{crown\_441,1}(rcis\_1) \), is shown in Figure 3-4.

---

**Figure 3-4: rcis\_sel cluster**

<table>
<thead>
<tr>
<th>Year</th>
<th>Dep</th>
<th>Crse #</th>
<th>Id</th>
<th>Instructor</th>
<th>Stud #</th>
<th>Studname</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall 89</td>
<td>CS</td>
<td>441</td>
<td>101</td>
<td>Fuller, J</td>
<td>041</td>
<td>Hu, Judy</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>109</td>
<td>Salona, Shyam</td>
<td>052</td>
<td>Smith, P</td>
<td>A</td>
</tr>
</tbody>
</table>
3.2.2 Normal Projection

Normal projection operation is used for chosing columns from a cluster. The result of a normal projection operation is always a relational cluster. A non-relational cluster can be constructed from the relational cluster by making use of the embed and unembed operations.

**Definition:** Let $c[S]$ be a cluster with schema $S$. The *normal projection* of $c[S]$ over $P$ where $P$ is defined subsequently, denoted as $\pi^p_n(c[S])$, is computed as follows:

$$P = \{ R_i \mid R_i \text{ is a zero order name and } R_i \in S \}$$

$$\pi^p_n[c] = \{ \rho_r \mid \forall R_i \in P \ \rho_r[R_i] = \rho_c[R_i] \text{ and } \rho_c \in c \}$$

For example, the normal projection of the rcis_sel cluster on attributes Sem, Yr, Dept, Crse#, Instruct, Studname, Gr is shown in Figure 3-5.

<table>
<thead>
<tr>
<th>Sem</th>
<th>Yr</th>
<th>Dept</th>
<th>Crse#</th>
<th>Instruct</th>
<th>Studname</th>
<th>Gr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Fuller, J</td>
<td>Hu, Judy</td>
<td>A</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Fuller, J</td>
<td>Hite, Lee</td>
<td>B</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Fuller, J</td>
<td>Smith, P</td>
<td>A</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Salona, Shyam</td>
<td>Hu, Judy</td>
<td>A</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Salona, Shyam</td>
<td>Hite, Lee</td>
<td>R</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Salona, Shyam</td>
<td>Smith, P</td>
<td>A</td>
</tr>
</tbody>
</table>

**Figure 3-5:** rcis_proj cluster
3.2.3 Renaming

Let $c[S]$ be a cluster with the schema $S$ and let $A$ be an attribute of $S$. Also, let $B$ be an attribute such that $B \notin (S - A)$ the renaming operation is defined as follows:

1. If $A$ is a zero-order name with the $\text{Domain}(A) = \text{Domain}(B)$. The renaming of $A$ to $B$ in the cluster $c[S]$, denoted by $\delta_{A \rightarrow B}(c[S])$, is a cluster with schema $S' = (S - A) \cup B$ such that the column $A$ of $S$ is replaced by the column $B$. The result cluster of the operation denoted by $r$ is derived as follows:

$$\delta_{A \rightarrow B}(c[S]) = \{ \rho_r \mid \rho_c \in c[S], \rho_r[S - A] = \rho_c[S - A] \land \rho_r[B] = \rho_c[A] \}$$

2. If $A = (A_1, A_2, \ldots, A_n)$ and $B = (B_1, B_2, \ldots, B_n)$ are higher order names such that $A$ and $B$ are compatible then, let $f$ be the compatibility function such that $B_j = f(A_i)$. The renaming of $A$ to $B$ in the cluster $c[S]$, denoted by $\delta_{A \rightarrow f}(c[S])$, is a cluster $r$ with schema $S' = (S - A) \cup B$ such that column $A$ in $S$ is replaced by column $B$. The cluster $r$ is derived as follows:

2.1 For all $A_i$ in $A$

$$\delta_{A \rightarrow f}(c[S]) = \{ \rho_r \mid \rho_c \in c[S] \land \rho_r[S - A_i] = \rho_c[S - A_i] \land \rho_r[f(A)] = \rho_c[A_i] \}$$

2.2 Replace name $A$ by name $B$ in $S'$. 
3.2.4 Expand

*Expand* is a macro that takes any zero-order or non-zero order name as input and generates an equivalent expanded version for that name. The expand is not an algebraic operation. Let $c[S]$ be a cluster with schema $S$ and let $A$ be a name in $S$. The expand operation is defined as follows:

1. If $A$ is a zero-order name then, $\text{expand}(A) = A$.

2. If $A = (A_1, A_2, ..., A_m)$ is a higher order name in the schema $S$ then,
   \[ \text{expand}(A) = A_1.A_2....A_m \]

For example, the result of $\text{expand}(\text{STUDINFO})$ is as follows:

\[ \text{expand}(\text{STUDINFO}) = \text{Studid}, \text{Studname}, \text{Date birth}, \text{Date adm} \]

3.2.5 Embed

The *embed* is an operator that alters the structural relationships amongst rules of the schema. This operator is similar to the nest operator defined by Roth [42] except that the nest operator is defined for the relational and hierarchical databases only.

**Definition:** Let $c[S]$ be a cluster with schema $S$. The embed operator, as the name implies, embeds a subset of attributes into the another subset of the attributes of the schema $S$. All cluster rows are suitably modified to reflect the change in the schema.
Let \( R_i \) be a non zero-order name in \( S \) and \( A = \{ A_1, A_2, \ldots, A_m \} \) be an attribute set which is a subset of attributes of \( R_i \). Also, the schema \( S \) does not have any rule with \( (A_1, A_2, \ldots, A_m) \) on its right hand side. Define a set of attributes \( B = \{ B_1, B_2, \ldots, B_m \} = R_i \setminus A \).

The embed operation is denoted by \( \eta(A, B, S_i') = c'[S'] \) where \( S' \) is generated as follows:

I. Define a rule \( R_i' = (B_1, B_2, \ldots, B_m, A = (A_1, A_2, \ldots, A_m)) \)

II. Also, define a new rule \( A = (A_1, A_2, \ldots, A_m) \)

III. Replace the rule for \( R_i \) in \( S \) by the new rule \( R_i' = (B_1, B_2, \ldots, B_m, A) \) in \( S \). Substitute the name \( S \) by \( S' \).

\( c' \) is generated as follows:

\[
c' = \left\{ \rho_r \mid \exists \rho_c \in c : \forall i \in \beta \exists \rho_c[B_i] = \rho_r[B_i] \land \rho_r[A] = \left\{ \exists \rho_c[A] \mid \exists \rho_c \in c : \forall i \in \beta \exists \rho_c[B_i] = \rho_c[B_i] \right\} \right\}
\]

For example, the result of embed operation, \( \eta(A, B, \text{rci} \text{. p} \text{roj}) \), where \( A = \{ \text{Studname, Gr} \} \) and \( B = \{ \text{Sem, Yr, Dept, crse#, Instruct} \} \) is shown in Figure 3-6.
3.2.6 Unembed

The unembed operation accomplishes the inverse of the embed operator. Simply stated, this operation flattens an embedded table. In other words, for a database cluster that has just one table embedded within the external table, the single unembed operation will produce a relational cluster.

**Definition:** Let $c[S]$ be a cluster with schema $S$. Let $R_i = (R_{i1}, R_{i2}, \ldots, R_{im})$ be a rule in $S$. Also, let $R_{ij} \in R_i$ be a non-zero order name such that $R_{ij} = (R_{ij1}, R_{ij2}, \ldots, R_{ijn})$ is a rule in $S$. The unembed operation $\phi(R_i, R_{ij}, c[S]) = c'[S']$ is defined as follows:

$S'$ is derived from $S$ by

**Case I.** $R_{ij}$ appears only on the right hand side of a rule $R_i$ in $S$.

i) Copy all the rules from $S$ to $S'$. 

<table>
<thead>
<tr>
<th>Sem</th>
<th>Yr</th>
<th>Dept</th>
<th>Crse#</th>
<th>Instruct</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Studname</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Fuller, J</td>
<td>Hu, Judy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Hite, Lee</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Smith, P</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Salona, Shyam</td>
<td>Hu, Judy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Hite, Lee</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Smith, P</td>
</tr>
</tbody>
</table>

Figure 3-6: rcis_embed cluster
ii) Modify the right hand side of the $R_i$ in $S$ to $R_i = (R_{i1}, R_{i2}, ..., R_{ij1}, R_{ij2}, R_{ij3}, ..., R_{ijn}, ..., R_{im})$.

iii) Drop the rule $R_{ij} = (R_{ij1}, R_{ij2}, ..., R_{ijn})$ from $S$.

Case II. $R_{ij}$ appears on the right hand side of a rule $R_j$ in $S$ such that $R_i \neq R_j$ and $R_j = (R_{j1}, R_{j2}, ..., R_{jn}, ..., R_{jm})$.

i) Copy all the rules from $S$ to $S'$

ii) Modify the right hand side of the $R_i$ in $S'$ to $R_i = (R_{i1}, R_{i2}, ..., R_{ij1}, R_{ij2}, R_{ij3}, ..., R_{ijn}, ..., R_{im})$.

iii) Replace $R_{ij}$ by $R_i$ in all the $R_j$ rules where $R_{ij}$ appears on the right hand side. Thus $R_j$ rule becomes $R_j = (R_{j1}, R_{j2}, ..., R_{jn}, ..., R_{jm})$.

iv) Drop the rule where $R_{ij}$ appears on the left hand side.

c' is derived from $c$ by

Let $t(R_{ij})$ denote a table row of $R_{ij}$ in $R_i$. Let $t_{pr}(R_{ij}, R_i)$ denote the parent row of $t(R_{ij})$ in $R_i$.

For each table row $t(R_{ij})$ generate a new table row, $t_r$, of $R_i$ as follows:

I. Let $t^* = t_{pr}(R_{ij}, R_i)$ be the parent row.

II. $\forall R_{ik} \in R_i \land R_{ik} \notin R_{ij}$ then $t_r[R_{ik}] = t^*[R_{ik}]$
III. \( \forall R_{ik} \in R_{ij} \ t_r[R_{ik}] = t_{ij}[R_{ik}] \)

IV. For all \( R_j \) such that \( R_{ij} \) appears on the right hand side and \( R_j \neq R_i \)

\[ t_{pr}(R_i, R_j) = t_{pr}(R_{ij}, R_j) \]

For example, the result of unembed operation, \( \phi(B, A, rcis_{emb}) \), is given in Figure 3-7.

<table>
<thead>
<tr>
<th>Sem</th>
<th>Yr</th>
<th>Dept</th>
<th>Crse#</th>
<th>Instruct</th>
<th>Studname</th>
<th>Gr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Fuller, J</td>
<td>Hu, Judy</td>
<td>A</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Fuller, J</td>
<td>Hite, Lee</td>
<td>B</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Fuller, J</td>
<td>Smith, P</td>
<td>A</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Salona, Shyam</td>
<td>Hu, Judy</td>
<td>A</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Salona, Shyam</td>
<td>Hite, Lee</td>
<td>B</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>Salona, Shyam</td>
<td>Smith, P</td>
<td>A</td>
</tr>
</tbody>
</table>

Figure 3-7: rcis_unemb cluster

3.2.7 Normalize

The normalize operation transforms any heterogeneous database cluster to a database cluster with relational schema.

**Definition:** Let \( c[S] \) be a cluster with schema \( S \). The operation normalize, \( N(c[S]) \), generates an equivalent cluster with a relational schema.

The algorithm to derive the normalized relational cluster \( r[S'] \) from the original cluster \( c[S] \) is as follows:
Step 1. Set $r[S'] = c[S]$.

Step 2. Repeat 
\{
  Decompose $S'$ in subsets $A$ and $B$ such that 
  \[ S' = A \cup B \] 
  and 
  $A$ is a set of all zero-order names in $S'$ and 
  $B$ is a set of all non-zero order names in $S'$.
  
  If $B$ is not empty then, 
  \[ \forall_{b_i \in B} \ r[S'] = N(unembed(S', B, c'[S'])) \]
\} until $B$ is an empty set

3.2.8 Union

The union operation is a binary operation on the cluster operands. The operands of an union operation are always compatible.

**Definition:** Let $c[S_1]$ and $d[S_2]$ be two compatible clusters. The *union* of the two compatible clusters $c \cup d$ is a cluster consisting of the rows belonging to either $c$ or $d$ or both. The result of the union operation $r[S]$ is derived as follows:

1. $S_1$ and $S_2$ are compatible so either $S_1 = S_2$ or generate $S_2'$ by renaming all the attributes in $(S_2 - S_1) \cap S_2$ to $(S_1 - S_2) \cap S_1$ such that $S_1 = S_2'$.

2. Let $P$ be the subset of the $S_1$ such that
case I. \( \exists \rho_c \in c \)
\[ \forall \rho_d \in d \ \exists \ A_i \ \exists \ \rho_c[A_i] \neq \rho_d[A_i] \text{ then,} \]
\[ c \cup d = \{ \rho_r \mid \rho_r = \rho_c \} \]

case II. \( \exists \ \rho_d \in d \)
\[ \forall \rho_c \in c \ \exists \ A_i \ \exists \ \rho_d[A_i] \neq \rho_c[A_i] \text{ then,} \]
\[ c \cup d = \{ \rho_r \mid \rho_r = \rho_d \} \]

case III. \( \exists \ \rho_c \in c \) and \( \exists \ \rho_d \in d \)
\[ \forall A_i \ \rho_d[A_i] = \rho_c[A_i] \text{ then,} \]
\[ c \cup d = \{ \rho_r \mid \forall A_i \ \rho_r[A_i] = \rho_c[A_i] \land \]
\[ \forall B_i \ \rho_r[B_i] = \rho_c[B_i] \cup \rho_d[B_i] \} \]

Whenever \( Q \) is an empty set, it is an usual union of relations.

For example, the result of union of rcis_1 and rcis_2 clusters denoted by \( \text{rcis\_union} = \text{rcis\_1} \cup \text{rcis\_2} \) is shown in Figure 3-8.
<table>
<thead>
<tr>
<th>Sem</th>
<th>Yr</th>
<th>Dept</th>
<th>Crse#</th>
<th>Id#</th>
<th>Instructor</th>
<th>Stud#</th>
<th>Studname</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall 89</td>
<td></td>
<td>CS</td>
<td>441</td>
<td>101</td>
<td>Fuller, J</td>
<td>042</td>
<td>Hu, Julia</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>109</td>
<td>Salona, Shyam</td>
<td>048</td>
<td>Hayte, Anne</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>451</td>
<td>104</td>
<td>Halex, Milt</td>
<td>110</td>
<td>Goff, Roger</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>112</td>
<td>Virkar, R</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>453</td>
<td>102</td>
<td>Gordon, Chris</td>
<td>099</td>
<td>Oza, Amish</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>049</td>
<td>Hite, Lee</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>451</td>
<td>104</td>
<td>Halex, Milt</td>
<td>110</td>
<td>Goff, Roger</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>112</td>
<td>Virkar, R</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>453</td>
<td>102</td>
<td>Gordon, Chris</td>
<td>099</td>
<td>Oza, Amish</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>049</td>
<td>Hite, Lee</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>444</td>
<td>111</td>
<td>Wakim, N</td>
<td>042</td>
<td>Hu, Steve</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>155</td>
<td>Waite, Terry</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>445</td>
<td>109</td>
<td>Hwang, Steve</td>
<td>113</td>
<td>Taylor, R</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Salona, Shyam</td>
<td>150</td>
<td>Wang, Paul</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>441</td>
<td>101</td>
<td>Fuller, J</td>
<td>052</td>
<td>Berbert, John</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100</td>
<td>Paripati, P</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>444</td>
<td>111</td>
<td>Wakim, N</td>
<td>084</td>
<td>Lee, Kyun-He</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>116</td>
<td>Shah, S</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>457</td>
<td>109</td>
<td>Hwang, Steve</td>
<td>035</td>
<td>Biden, Tom</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Salona, Shyam</td>
<td>150</td>
<td>Wang, Paul</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CS</td>
<td>458</td>
<td>101</td>
<td>Fuller, J</td>
<td>084</td>
<td>Lee, Kyun-He</td>
<td>A</td>
</tr>
</tbody>
</table>

Figure 3–8: rcis_union cluster

3.2.9 Intersection

The intersection operation of clusters accepts two compatible clusters as input and produces a cluster.
Definition: Let \( c[S_1] \) and \( d[S_2] \) be two compatible clusters. The intersection of the two compatible clusters, \( c \cap d \), is a cluster consisting of the rows which belong to both clusters \( c \) and \( d \). The result of the intersection operation is derived as follows:

1. \( S_1 \) and \( S_2 \) are compatible so either \( S_1 = S_2 \) or generate \( S_2' \) by renaming all the attributes in \((S_2 - S_1) \cap S_2\) to \((S_1 \cdot S_2) \cap S_1\) such that \( S_1 = S_2' \).

2. Let \( P \) be the subset of the \( S_1 \) such that

The result of intersection is generated as follows:

\[
\exists \ \rho_c \in c, \ \exists \ \rho_d \in d \ \exists \ \forall A_i, \ \rho_c[A_i] = \rho_d[A_i] \ \text{and}
\forall B_i, \ \rho_c[B_i] \cap \rho_d[B_i] \neq \emptyset \text{ then,}
\exists \ \rho_r \in P \ | \ \forall A_i, \ \rho_r[A_i] = \rho_c[A_i] \ \wedge
\forall B_i, \ \rho_r[B_i] = \rho_c[B_i] \cap \rho_d[B_i]}
\]

The relational intersection is a special case of above definition where \( P \) is an empty set.

For example, the result of the intersection of the \texttt{rcis\_1} and \texttt{rcis\_2} cluster denoted by \texttt{rcis\_intersect = rcis\_1 \cap rcis\_2} is shown in Figure 3-9.
3.2.10 Difference

The difference operation of clusters is a binary operation, that accepts two compatible clusters as arguments. The result of the operation is a cluster.

**Definition:** Let $c[S_1]$ and $d[S_2]$ be two compatible clusters. The *difference* of the two clusters, written as $c - d$, retains those rows of the cluster $c$ that are not in the cluster $d$. It is defined as follows:

1. $S_1$ and $S_2$ are compatible so either $S_1 = S_2$ or generate $S_2'$ by renaming all the attributes in $(S_2 - S_1) \cap S_2$ to $(S_1 - S_2) \cap S_1$ such that $S_1 = S_2'$.

2. Let $P$ be the subset of the $S_1$ such that

Result of difference operation is generated as follows:

Case I.
Case II. \( \exists \rho_c \in c \ \forall \rho_d \ \exists A_i \ (\rho_c[A_i] \neq \rho_d[A_i]) \) then,

\[ c - cl = \{ \rho_r \mid \rho_r = \rho_c \} \]

Whenever \( Q \) is an empty set, it is the usual difference of relations.

For example, the difference of rcis_1 and rcis_2 clusters, rcis_diff = rcis_1 - rcis_2, is a cluster shown in Figure 3-10.

<table>
<thead>
<tr>
<th>Registrar</th>
<th>Courses</th>
<th>Student</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dept</td>
<td>Crse#</td>
</tr>
<tr>
<td>Fall 89</td>
<td>CS</td>
<td>441</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CS</td>
<td>442</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CS</td>
<td>443</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spring '89</td>
<td>CS</td>
<td>444</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CS</td>
<td>457</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CS</td>
<td>458</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3-10: rcis_diff cluster

3.2.11 Cross Product

The cross product of clusters is a binary operation. The cross product requires that the schemata of the two clusters do not have qualified duplicate names. The result of the operation is a cluster.
**Definition:** Let $c[S_1]$ and $d[S_2]$ be two clusters. Also, $S_1 \cap S_2 = \emptyset$. The *cross product* of the two clusters, $c \times d$, is defined by

$$c[S_1] \times d[S_2] = r[S']$$

where

$S'$ is derived from $S_1$ and $S_2$ by the following algorithm:

1. Copy all rules from $S_1$ to $S'$.
2. Copy all rules from $S_2$ to $S'$.
3. Add a new rule $S' = (\text{expand}(S_1), \text{expand}(S_2))$
4. Drop the rules corresponding to $S_1$ and $S_2$ on the left hand side.

$r$ is derived from $c$ and $d$ by following algorithm

For each $\rho_c \in c$

For all $\rho_d \in d$

For all $A_i \in S'$

$$If \ A_i \in S_1 \ \ rho_r[A_i] = rho_c[A_i]$$

$$If \ A_i \in S_2 \ \ rho_r[A_i] = rho_d[A_i]$$
3.2.12 Normal Natural Join

The operation of normal natural join is an equijoin of two clusters on all the common attributes in their normalized schema. The normal natural join retains only one of such duplicate columns.

**Definition:** Let \( c[S_1] \) and \( d[S_2] \) be two clusters with schemata \( S_1 \) and \( S_2 \) respectively. The *normal natural join*, \( r[S'] = c \text{ NJN } d \), is defined as follows:

Let

Let

Let \( Z = X \cap Y \) then,

\[ S' = (X, Z, Y) \]

\( r \) is derived from \( c' \) and \( d' \) as follows:

\[ r' = \{ \rho_r : \exists \rho_c \in c, \rho_d \in d \mid \forall _{Z_i} \rho_c[Z_i] = \rho_d[Z_i] ; \rho_r[X] = \rho_c[X] \wedge \rho_r[Y - Z] = \rho_d[Y - Z] \} \]

For example, the result of rcis_1 NJN studinfo is shown in Figure 3-11.
<table>
<thead>
<tr>
<th>Sem</th>
<th>Yr</th>
<th>Dept</th>
<th>Cnrf#</th>
<th>Id#</th>
<th>Instruct</th>
<th>Stuid</th>
<th>Studname</th>
<th>Gr</th>
<th>Date_birth</th>
<th>Date_adm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>101</td>
<td>Fuller,J</td>
<td>041</td>
<td>Hu,Judy</td>
<td>A</td>
<td>June,22,1964</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>101</td>
<td>Fuller,J</td>
<td>049</td>
<td>Hite,Lee</td>
<td>B</td>
<td>Feb,28,1963</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>101</td>
<td>Fuller,J</td>
<td>052</td>
<td>Smith,P</td>
<td>A</td>
<td>Jan,04,1963</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>109</td>
<td>Salaria,S</td>
<td>041</td>
<td>Hu,Judy</td>
<td>A</td>
<td>June,22,1964</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>109</td>
<td>Salaria,S</td>
<td>049</td>
<td>Hite,Lee</td>
<td>B</td>
<td>Feb,28,1963</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>441</td>
<td>109</td>
<td>Salaria,S</td>
<td>052</td>
<td>Smith,P</td>
<td>A</td>
<td>Jan,04,1963</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>442</td>
<td>103</td>
<td>Green,Jim</td>
<td>110</td>
<td>Goff,Roger</td>
<td>A</td>
<td>Aug,30,1965</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>442</td>
<td>103</td>
<td>Green,Jim</td>
<td>112</td>
<td>Virker,R</td>
<td>B</td>
<td>Oct,02,1959</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>442</td>
<td>103</td>
<td>Green,Jim</td>
<td>052</td>
<td>Smith,P</td>
<td>A</td>
<td>Jan,04,1963</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>443</td>
<td>102</td>
<td>Gordon,C</td>
<td>099</td>
<td>Oza,Amita</td>
<td>A</td>
<td>Nov,23,1964</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>443</td>
<td>102</td>
<td>Gordon,C</td>
<td>049</td>
<td>Hite,Lee</td>
<td>B</td>
<td>Feb,28,1963</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Fall</td>
<td>89</td>
<td>CS</td>
<td>443</td>
<td>102</td>
<td>Gordon,C</td>
<td>115</td>
<td>Salgam,R</td>
<td>A</td>
<td>May,14,1952</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Spring</td>
<td>89</td>
<td>CS</td>
<td>457</td>
<td>109</td>
<td>Salaria,S</td>
<td>150</td>
<td>Wang,Paul</td>
<td>A</td>
<td>July,14,1966</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Spring</td>
<td>89</td>
<td>CS</td>
<td>457</td>
<td>109</td>
<td>Salaria,S</td>
<td>052</td>
<td>Smith,Paul</td>
<td>A</td>
<td>Jan,04,1963</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Spring</td>
<td>89</td>
<td>CS</td>
<td>458</td>
<td>101</td>
<td>Fuller,J</td>
<td>100</td>
<td>Peripati,P</td>
<td>A</td>
<td>Dec,25,1964</td>
<td>Sep,05,1988</td>
</tr>
<tr>
<td>Spring</td>
<td>89</td>
<td>CS</td>
<td>458</td>
<td>101</td>
<td>Fuller,J</td>
<td>115</td>
<td>Shah,S</td>
<td>A</td>
<td>Apr,23,1965</td>
<td>Sep,05,1988</td>
</tr>
</tbody>
</table>

Figure 3-11: rcis_studinfo_join cluster

3.3 Conclusion

The heterogeneous data model used in this work is a suitable choice for storing complex objects compared to the relational model. This data model allows integration of the data stored in hierarchical and network data models. The query languages developed for this data model, GSQL and GCALC, permit uniform access to the data stored in the relational as well as the non-relational data models. However, these query languages are not suitable for the query optimization purposes. The cluster algebra has been defined to fill this void. The cluster algebra operators perform one single step of the query at a time. This provides greater flexibility of rearranging and restructuring the sequence of operators needed to answer a query. Various extensions of relational algebra proposed [25,42] for the recursive extensions of relational model usually encompass the data stored in PNF [42] or similar formats. These extensions
have the scope of the relational and the hierarchical data models. The cluster algebra proposed here embodies the uniform access mechanism for the data stored in the relational, hierarchical and the network data models.
4 Query Translation: Virtual to Conceptual Clusters

4.1 Introduction

The conceptual clusters are the clusters that are physically stored in a database. The virtual clusters are windows defined over the conceptual clusters. The data for virtual clusters is not directly stored on a physical media, instead a mapping between a virtual cluster and the conceptual clusters is maintained by the system. Any access or update on a virtual cluster is translated into a set of equivalent operations on the conceptual clusters.

In a system of heterogeneous databases, the database fragments may be stored in different data models. Since, the fragment databases are developed independently, the fragments may have attribute name, table name and structural conflicts. The integration of the fragment databases is done by defining virtual clusters over the fragments. The schema mapping architecture of Figure 2-2 is used for defining the unified global schema for the heterogeneous databases in the DAVID system. The queries expressed over global schema may contain references to the virtual clusters.

Additionally, a conceptual cluster provides only one view of the data. In general, the single view of the data may not suffice for a diverse set of users and applications. The need for having multiple views or alternate views for the same conceptual schema may manifest itself due to variety of reasons. Some of the reasons are discussed subsequently.
The security considerations may require that certain data be made "invisible" to a set of users. The need may be illustrated by a library database given as follows:

\[
\text{Holdings} = (\text{Access_no, Isbn_no, Title, Authors, Price, Patron_id, Due_date})
\]

\[
\text{Authors} = (\text{Last_name, First_name, Middle_Init})
\]

\[
\text{Patrons} = (\text{Patron_id, Last_name, First_name, Address})
\]

The library patrons have need to access the library Holdings data set and its details such as the accession number, the authors, the title and the due date of the book in the library but should not be allowed to see the patron_id and other such details of a borrower. Thus, the data item patron_id in Holdings data set needs to be made "invisible" to all the patrons of library. This can be achieved by building an alternate view of the Holdings cluster, that is described as follows:

\[
\text{P_Holdings} = (\text{Access_no, Isbn_no, Title, Authors, Price, Due_date})
\]

In the cluster algebra the above view can be defined by the following expression:

\[
\pi_{\text{Access_no, Isbn_no, Title, Authors, Price, Due_date}}(\text{Holdings})
\]

The above view of the Holdings cluster expressed in the GSQL is given as follows:
Create view P_holdings
Select Access_no, Isbn_no, Title, Authors, Price, Due_date
From Holdings;

Apart from the security considerations, users may have conflicting needs or preference for a particular way of viewing the data. A solution to this problem can be provided by creating a personalized view over a collection of conceptual clusters. The personalized view being the one that best serves the user requirements and ease of the use. To illustrate the point, let us assume that a university maintains its Courses, Instructor and Student information in a hierarchical, rcis_1[RCIS], cluster. A group of departments prefer to view the same data as a relational cluster. This problem can be solved by creating a virtual cluster or a view of the data called registrar.

In cluster algebra the expression for doing so is given as follows:

\[ \pi_{Sem., Yr., Dept., Courses} (rcis_1) \]

The same view can be expressed in GSQl as follows:

Create view registrar
Select Sem, Yr, Dept, Crse#, Id#, Instruct, Studid, Studname, Gr
From rcis_1;

The term view is also used for the virtual cluster in this work. An arbitrary number of virtual clusters on top of the conceptual clusters can be defined in a system of heterogeneous databases.
The virtual clusters also provide the function of conceptual level data-independence. Any change in the conceptual model does not impact the users of the virtual clusters unless any data item being referred to in the virtual cluster has been modified or deleted. This concept allows dynamic modification of the conceptual model without any impact on the application programs. Also, anytime a conceptual schema is modified the applications written on the old conceptual model can be supported by defining a virtual cluster with the old schema. This allows us to maintain the old software in dynamic environment.

The virtual clusters may also be used to present the partitioned view of the data. The concept of partitioning has already been discussed in preceding chapters. For example, a university with several campuses may maintain all the registrar information in a single cluster whose schema is as follows:

```sql
define cluster rcis_campus ( 
    registrar table ( 
        sem char[10]; 
        yr char[4]; 
        campus char[20]; 
        courses table ( 
            dept char[2]; 
            crse# char[4]; 
            instructor table ( 
                id# char[10]; 
                instruct char[30]; 
            ); 
            student table ( 
                studid char[10]; 
                studname char[30]; 
                gr char; 
            ); 
        ); 
    ); 
); 
```
Each individual campus administrator is interested in only the information about their own campus. This situation can be resolved by defining virtual clusters or views that limit the access of information to one campus only.

For example, let Blacksburg, Dahlgren and Telestar be the three campuses for an university. The partitioned virtual clusters for each campus can be defined as follows:

\[ rcis\_blackburg = \sigma_{\text{campus}=\text{Blacksburg}}(rcis\_campus) \]

\[ rcis\_dahlgren = \sigma_{\text{campus}=\text{Dahlgren}}(rcis\_campus) \]

\[ rcis\_telestar = \sigma_{\text{campus}=\text{Telestar}}(rcis\_campus) \]

The \( rcis\_blackburg \) cluster limits the access of \( rcis\_campus \) data to the Blacksburg campus only. Similarly, the other two virtual clusters provide access to the information about Dahlgren and Telestar campuses only.
4.2 Virtual Cluster Definition

The virtual clusters are defined by the mechanism of "mappings" between the virtual and conceptual clusters. The mapping consists of "interpretation" and a set of operation simulators. The formal way to define the mapping has already been described in the preceding chapters.

The virtual clusters can also be defined using any of the query languages discussed in this work: the GSQL, the GCALC and the Cluster Algebra. In the cluster algebra, a sequence of operations that form a valid cluster algebra expression is used for defining a virtual cluster. The result of a cluster algebra expression is a cluster. The syntax to define a virtual cluster in the cluster algebra is described as follows:

\[
\text{<View\_name\>} := V(\text{<alg\_exp>})
\]

where:

\[
\text{<View\_name\>} : \quad \text{A virtual cluster name}
\]

\[
\text{<alg\_exp\>} : \quad \text{A valid cluster algebra expression over the variables of conceptual clusters. The syntax for defining the valid cluster algebra expressions is presented later in this chapter.}
\]

In heterogeneous database management systems the global schema is built by defining virtual clusters to resolve inter-fragment conflicts. Thus, queries in a system
for heterogeneous databases may contain references to the virtual clusters. In order to process these queries, all the references to the virtual clusters have to be translated to the conceptual clusters.

4.3 Query Translation: Virtual to Conceptual Clusters

An algorithm to translate queries expressed over the virtual clusters to those expressed over the conceptual clusters is described here. The translation algorithm replaces occurrences of the virtual clusters from the original query by the conceptual clusters and modifies it to generate a semantically equivalent query. The translation algorithm, for queries expressed in the cluster algebra, is described subsequently.

4.3.1 Cluster Algebra Queries

A cluster algebra expression (query) over the uniformized schema may involve virtual clusters. The virtual clusters are defined using the "mapping". The "mapping" is also a cluster algebra expression over other virtual and conceptual clusters. In order to answer a query expressed over virtual clusters, the original query has to be transformed into an equivalent query with all references to the virtual clusters replaced by the conceptual clusters.

In order to present the algorithm, for translating a query expressed over virtual clusters to an equivalent query expressed over the conceptual cluster, the following notation and definitions are used.
The grammar for generating the valid cluster algebra expressions is defined as follows:

\[
\begin{align*}
\text{AE} & \rightarrow \ <\text{cluster\_name}> \\
\text{AE} & \rightarrow \ (\text{AE}) \\
\text{AE} & \rightarrow \ \text{unary AE} \\
\text{AE} & \rightarrow \ \text{AE binary AE} \\
\text{unary} & \rightarrow \ \{\pi, \ \sigma, \ \eta, \ \mu\} \\
\text{binary} & \rightarrow \ \{\cup, \ \cap, \ -, \ N\bar{N}, \ \alpha\}
\end{align*}
\]

For example, \(\pi_{\text{studname}, \ dob}(\sigma_{\text{sem\_fall}}(\text{registrar} \ N\bar{N} \ studinfo))\) is a valid cluster algebra query as per the grammar. The parse tree for this query using the above grammar is given in Figure 4-1.

In order to facilitate the presentation of the query translation algorithm, the following notation is introduced.

\(Q\) : A query expressed by the user in the cluster algebra

\(P_Q\) : A parse tree for the query \(Q\)

\(C_i\) : A virtual or conceptual cluster

\(N_i\) : An \(i^{\text{th}}\) node of a parse tree
Figure 4-1: Parse tree for a cluster algebra expression

\( T_{c_i} \): Type of the cluster \( C_i \). It is either virtual or conceptual.

\( V(C_i) \): The mapping expression for the virtual cluster \( C_i \).

4.3.2 Query Translation Algorithm

A virtual to conceptual query translation algorithm for the queries expressed in the cluster algebra is presented here. The algorithm assumes that the "mappings"
for the virtual clusters referenced in the query are also expressed in the cluster algebra. In the first step of the algorithm, the query is parsed to generate the parse tree data structure. In the main loop of the algorithm, the parse tree for the query is scanned for all the cluster references appearing at the leaf nodes of the parse tree. Each of the cluster appearing in the leaf node is examined for its type. The type of a cluster can be either conceptual or virtual. For the virtual clusters the algorithm retrieves the "mapping" expression and generates a parse tree for the mapping expression. The leaf node with the virtual cluster reference is replaced by the root of the newly parsed tree for the "mapping" expression. The main loop iterates until all the references to the virtual clusters in the original query has been replaced by the references to the conceptual clusters. The resulting parse tree at end of the main loop labeled as step 3 has all the leaf nodes consisting of conceptual clusters. The final step, labeled as 4, of the algorithm generates the query expression from the parse tree. Since the final parse tree is free of any occurrences of the virtual clusters, the query expression generated from the parse tree will also be free of any references to the virtual clusters.
Virtual_to_conceptual(Q: cluster algebra expression)

1. Parse the query Q and let P_q be the parse tree generated for the query.
2. Set flag = TRUE
3. While (flag)
   begin
   flag = FALSE;
   Generate a list L containing all leaf nodes of parse tree P_q;
3.1. For each leaf node N_i in the list L
   begin
   Let C_i be the name of cluster stored in node N_i;
   Retrieve type $T_{C_i}$ of the cluster C_i;
   if $T_{C_i}$ is virtual
   begin
   Let AE_i = V(C_i) be the "mapping" expression;
   Let $P_{AE_i}$ be the parse tree for AE_i;
   replace node N_i by the parse tree $P_{AE_i}$;
   set flag = TRUE;
   end
   end
   end
4. Generate the cluster algebra expression from the parse tree P_q;
The above algorithm replaces all the virtual cluster occurrences in the parse tree $P_q$ by the conceptual clusters. The result query expressed over the conceptual clusters can be generated by "un-parsing" the resulting parse tree of this algorithm. The algorithm scans all the clusters at leaf nodes of the parse tree of the query and examines the type of the cluster. A leaf node with the virtual clusters is replaced by the parse tree for its mapping expression. The algorithm iterates until all the virtual cluster occurrences have been eliminated from the parse tree of the query.

For example, let IUE50 be a virtual cluster defined by the expression given as follows:

$$V(IUE50) = \sigma_{class>50}(Iuelog) \quad NJN \quad Pilog$$

The instances of Iuelog and Pilog clusters have been shown in Figure 5.4 and 5.8 respectively. The parse tree of the cluster algebra expression for the view IUE50 is shown in Figure 4-2.
Figure 4–2: Parse tree for $V(\text{UE50})$

A query $Q$ to find the image names of all the Iue observations for class 50 and higher objects which were also observed by the Iras mission is expressed in cluster algebra as follows:

$$Q = \pi_{\text{image}_-\text{name}}(\text{UE50, NJN, Iraspsc})$$

The parse tree $P_q$ for the query $Q$ is shown in Figure 4-3.
The parse tree $P_q$ has IUE50 and Iraspce clusters in its leaves. The IUE50 is a virtual cluster. Following the algorithm presented above the node is replaced by the parse tree of the expression for $V(IUE_{50})$. The resulting parse tree of the above operation is shown in Figure 4-4.

The parse tree of Figure 4-4 has Iuelog, Pilog and Iraspce clusters in its leaf nodes. All the above clusters are conceptual clusters, hence the algorithm terminates. The parse tree of Figure 4-4 represents an equivalent query expressed over conceptual clusters. The equivalent query represented by the parse tree is as follows:
\[ \pi_{\text{image}_\text{name}}((\sigma_{\text{class}>50}(\text{luelog NJN PiLog})) \text{ NJN Iraspqc}) \]

**Figure 4-4: Parse tree for the result query**
4.4 Conclusions

The query translation algorithm described in this chapter was implemented as a part of the query processing module of the DAVID system. The algorithm translates queries expressed in the cluster algebra over the virtual clusters to equivalent queries that are free of any occurrences of the virtual views. The algorithm assumes its input queries are free of any nested and aggregate functions. The nested queries and queries with aggregate functions are referred to as complex queries. The complex queries are translated to the basic form by pre-processing modules of the DAVID system. The output of this algorithm is also a basic query which is free of all instances of virtual clusters. In chapter 5, a technique to translate a basic query to primitive database operations will be described.
5 Query Decomposition in Heterogeneous Distributed Database Management Systems

The query processing in a heterogeneous distributed database management system poses an additional set of problems compared to the relational distributed database management systems. The three most important issues with regard to the query processing in a heterogeneous database management system are discussed here. First, the traditional relational algebra operators can not be used directly on the heterogeneous databases. This issue was addressed by defining the cluster algebra in the preceding chapters. The cluster algebra is an extension of the relational algebra and applies uniformly to the heterogeneous databases as the relational algebra applies to the relational databases.

The second issue consists of the size estimation of all the temporary results clusters to answer the query expressed over heterogeneous databases. The size estimation issue for the heterogeneous databases is addressed by developing a non-parametric estimation technique presented later in this chapter. This technique tracks the attribute values in the data structures defined for the HDDBMSs and assumes that the query decomposition strategy is determined prior to the execution of any intermediate operations.

The final issue of efficiently accessing data in a HDDBMS is addressed by formulating and implementing heuristic algorithms that determine good query processing strategies that allow access to the information in a timely and cost effective fashion. Given that the query processing problem for distributed DBMSs is
NP-hard[30], one can formulate and implement heuristic algorithms that find good solutions but not necessarily globally optimal solutions. Specifically, the purpose of the query decomposition process for a HDDBMS is to partition a query expressed in terms of the actual underlying databases into a set of sub-queries. Subsequently, it selects the best strategy for sequencing the sub-query operations and their corresponding processing sites. 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 issue of efficient query processing and temporary size estimation for the heterogeneous databases are addressed in this chapter. In order to illustrate the algorithms presented here, a sample of heterogeneous distributed databases from the Astrophysics Data System are described in the following section.

5.1 A Sample Heterogeneous Distributed Database System

For the exposition purposes of this work, four data centers from the Astrophysics Data System (ADS) network were selected. These are depicted in Figure 5-1. The Astrophysics Data System consists of several distributed and heterogeneous databases. The main objective of the system is to provide uniform access to the astrophysical data regardless of it’s location, data model and the storage format.
5.1.1 IUE Regional Data Analysis Facility (IUE-RDAF)

The International Ultraviolet Explorer - Regional Data Analysis Facility (IUE-RDAF) maintains and manages observations made by the IUE satellite. The data objects maintained by the IUE-RDAF include among others the observation catalog, the raw images, the photometrically corrected images, the line-by-line spectra, the merged low dispersion spectra, the high dispersion spectra, the proposals and the descriptions of observations.

The Catalog of all observations (Iuelog)

Iuelog is the catalog of all the observations made by the IUE mission. The catalog contains the summary information about the observations. The summary information
is made up of the camera name (cam), the sequence number of the camera operations (img), the location of the object described by the right ascension (ra) and declination (dec), the time of observation (time), the exposure time (exp), the aperture size (ap), the class of object (cl) and the comment. The catalog keeps track of all observations and assists scientists in locating the objects of interest. For example, a scientist interested in visualizing all the galaxies lying in the declination band of 0 to +10 degrees can issue a restriction on the catalog specifying this as a condition. The result of this operations is a subset of the catalog, that can further be used for locating and visualizing images.

At IUE-RDAF the DBMS used for storing the Iuelog table is Oracle. The definition of this table in oracle is shown in Figure 5-2.

```
cREATE TABLE IUELLOG (  
  OBJECTID    CHAR[10];  
  CAM         CHAR[3];   
  IMG         INT;       
  RA          INT;       
  DEC         INT;       
  TIME        INT;       
  EXP         INT;       
  APERTURE    INT;       
  CLASS       INT;       
  COMMENT     CHAR[20];  
);  
```

Figure 5-2: Iuelog Table Definition in Oracle
The database table definition shown in Figure 5-2 is installed into the DAVID system using the resident primitive processor. The integrated schema definition of the above table in DAVID is shown in Figure 5-3. Also, Figure 5-4 shows an instance of the Iuelog table.

Define cluster IUE.iuelog (  
    objectid char[10];  
    cam char[3];  
    img int;  
    ra int;  
    dec int;  
    time int;  
    exp int;  
    aperture int;  
    class int;  
    comment char[20];  
) store as oracle(Iuelog);  

Figure 5-3: Definition of the Iuelog Table in the DAVID System

The Raw Image Catalog (Rilog)

The raw images are the images, taken by the vidicon camera system of the IUE mission, of points in the space. A point in the space is described by the RA and DEC coordinates in the Iuelog. For each image, a row in the Iuelog table is recorded. There may be several images for the same point in the space.
<table>
<thead>
<tr>
<th>Objid</th>
<th>Cam</th>
<th>img</th>
<th>Ra</th>
<th>Dec</th>
<th>Time</th>
<th>Exp</th>
<th>Ap</th>
<th>Cl</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>or_Nebul</td>
<td>LWP</td>
<td>1</td>
<td>2988</td>
<td>-1950</td>
<td>1255</td>
<td>9600</td>
<td>2</td>
<td>72</td>
<td>Orions Nebulae</td>
</tr>
<tr>
<td>or_Nebul</td>
<td>LWP</td>
<td>2</td>
<td>2988</td>
<td>-1950</td>
<td>1255</td>
<td>9600</td>
<td>2</td>
<td>72</td>
<td>Orions Nebulae</td>
</tr>
<tr>
<td>jupiter</td>
<td>LWP</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0920</td>
<td>9600</td>
<td>2</td>
<td>3</td>
<td>Scale = 3</td>
</tr>
<tr>
<td>m42area</td>
<td>LWP</td>
<td>4</td>
<td>2996</td>
<td>-1962</td>
<td>1839</td>
<td>2400</td>
<td>1</td>
<td>72</td>
<td>Original field</td>
</tr>
<tr>
<td>nova_vul</td>
<td>LWP</td>
<td>5</td>
<td>1097</td>
<td>7716</td>
<td>1112</td>
<td>9600</td>
<td>2</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>moon</td>
<td>LWP</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1730</td>
<td>9600</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>crabneb</td>
<td>LWP</td>
<td>7</td>
<td>2990</td>
<td>-1954</td>
<td>2230</td>
<td>2400</td>
<td>1</td>
<td>72</td>
<td>Crab Nebulae</td>
</tr>
<tr>
<td>or_Nebul</td>
<td>SWP</td>
<td>8</td>
<td>2988</td>
<td>-1950</td>
<td>0255</td>
<td>9600</td>
<td>2</td>
<td>72</td>
<td>Orions Nebulae</td>
</tr>
<tr>
<td>jupiter</td>
<td>SWP</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>1120</td>
<td>9600</td>
<td>2</td>
<td>3</td>
<td>Scale = 3</td>
</tr>
<tr>
<td>m42area</td>
<td>SWP</td>
<td>10</td>
<td>2996</td>
<td>-1962</td>
<td>2239</td>
<td>2400</td>
<td>1</td>
<td>72</td>
<td>Original field</td>
</tr>
<tr>
<td>nova_vul</td>
<td>SWP</td>
<td>11</td>
<td>1097</td>
<td>7716</td>
<td>1412</td>
<td>9600</td>
<td>2</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>moon</td>
<td>SWP</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0830</td>
<td>9600</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>crabneb</td>
<td>SWP</td>
<td>13</td>
<td>2990</td>
<td>-1954</td>
<td>2230</td>
<td>2400</td>
<td>1</td>
<td>72</td>
<td>Crab Nebulae</td>
</tr>
<tr>
<td>or_Nebul</td>
<td>Fes</td>
<td>14</td>
<td>2988</td>
<td>-1950</td>
<td>1055</td>
<td>9600</td>
<td>2</td>
<td>72</td>
<td>Orions Nebulae</td>
</tr>
<tr>
<td>jupiter</td>
<td>Fes</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>1128</td>
<td>9600</td>
<td>2</td>
<td>3</td>
<td>Scale = 3</td>
</tr>
<tr>
<td>m42area</td>
<td>Fes</td>
<td>16</td>
<td>2996</td>
<td>-1962</td>
<td>2039</td>
<td>2400</td>
<td>1</td>
<td>72</td>
<td>Original field</td>
</tr>
<tr>
<td>nova_vul</td>
<td>Fes</td>
<td>17</td>
<td>1097</td>
<td>7716</td>
<td>0412</td>
<td>9600</td>
<td>2</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>moon</td>
<td>Fes</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0730</td>
<td>9600</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>crabneb</td>
<td>Fes</td>
<td>19</td>
<td>2990</td>
<td>-1954</td>
<td>1030</td>
<td>2400</td>
<td>1</td>
<td>72</td>
<td>Crab Nebulae</td>
</tr>
<tr>
<td>or_Nebul</td>
<td>SWP</td>
<td>20</td>
<td>2988</td>
<td>-1950</td>
<td>1955</td>
<td>9600</td>
<td>2</td>
<td>72</td>
<td>Orions Nebulae</td>
</tr>
<tr>
<td>jupiter</td>
<td>SWP</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0720</td>
<td>9600</td>
<td>2</td>
<td>3</td>
<td>Scale = 3</td>
</tr>
<tr>
<td>m42area</td>
<td>SWP</td>
<td>22</td>
<td>2996</td>
<td>-1962</td>
<td>2339</td>
<td>2400</td>
<td>1</td>
<td>72</td>
<td>Original field</td>
</tr>
<tr>
<td>nova_vul</td>
<td>SWP</td>
<td>23</td>
<td>1097</td>
<td>7716</td>
<td>0312</td>
<td>9600</td>
<td>2</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>moon</td>
<td>SWP</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>0230</td>
<td>9600</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>crabneb</td>
<td>SWP</td>
<td>25</td>
<td>2990</td>
<td>-1954</td>
<td>1130</td>
<td>2400</td>
<td>1</td>
<td>72</td>
<td>Crab Nebulae</td>
</tr>
</tbody>
</table>

Figure 5-4: An instance of Iuelog cluster
The Rilog table contains the names of all the image files stored in the system. It is made up of the camera name (camera), the unique image number (img) and the file containing the image (imagename). The raw images catalog is a relational table. The schema of Rilog catalog is described in Figure 5-5. An instance of the Rilog cluster is shown in Figure 5-6.

```
Define cluster IUEIM.Rilog (
   camera char[3];
   img int;
   imagename char[20];
) store as Ingres(Rilog);
```

Figure 5-5: The unified schema of the Rilog table

The Photometrically Corrected Images Log (Pilog)

In order to get quantitatively meaningful data from the IUE raw images, the raw images are corrected for the instrumental effects. The correction compensates for the geometrical distortions, photometric non-linearities and non-uniformities.

The Pilog table contains the camera name (camera), the unique image number (img) and the file name containing the image (imagename). The photometrically corrected images catalog (Pilog) is a relational table whose schema is described in Figure 5-7. An instance of the Pilog cluster is shown in Figure 5-8.
<table>
<thead>
<tr>
<th>Camera</th>
<th>img</th>
<th>Image_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>LWP</td>
<td>1</td>
<td>LWP11ria.ueim</td>
</tr>
<tr>
<td>LWP</td>
<td>2</td>
<td>LWP21ria.ueim</td>
</tr>
<tr>
<td>LWP</td>
<td>3</td>
<td>LWP32rib.ueim</td>
</tr>
<tr>
<td>LWP</td>
<td>4</td>
<td>LWP41ria.ueim</td>
</tr>
<tr>
<td>LWP</td>
<td>5</td>
<td>LWP51ria.ueim</td>
</tr>
<tr>
<td>LWP</td>
<td>6</td>
<td>LWP62ria.ueim</td>
</tr>
<tr>
<td>LWP</td>
<td>7</td>
<td>LWP71ria.ueim</td>
</tr>
<tr>
<td>SWP</td>
<td>8</td>
<td>SWP81rib.ueim</td>
</tr>
<tr>
<td>SWP</td>
<td>9</td>
<td>SWP91ria.ueim</td>
</tr>
<tr>
<td>SWP</td>
<td>10</td>
<td>SWP102ria.ueim</td>
</tr>
<tr>
<td>SWP</td>
<td>11</td>
<td>SWP111rib.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>12</td>
<td>SWP122ria.ueim</td>
</tr>
<tr>
<td>SWP</td>
<td>13</td>
<td>SWP131ria.dat</td>
</tr>
<tr>
<td>FES</td>
<td>14</td>
<td>FES142rif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>15</td>
<td>FES152rif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>16</td>
<td>FES161ria.dat</td>
</tr>
<tr>
<td>FES</td>
<td>17</td>
<td>FES172rif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>18</td>
<td>FES181rif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>19</td>
<td>FES192rif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>20</td>
<td>FES202rif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>21</td>
<td>SWP212ria.ueim</td>
</tr>
<tr>
<td>SWP</td>
<td>22</td>
<td>SWP222ria.ueim</td>
</tr>
<tr>
<td>SWP</td>
<td>23</td>
<td>SWP231rib.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>24</td>
<td>SWP241rib.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>25</td>
<td>SWP252ria.ueim</td>
</tr>
<tr>
<td>FES</td>
<td>26</td>
<td>FES262ria.ueim</td>
</tr>
<tr>
<td>FES</td>
<td>27</td>
<td>FES272rif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>28</td>
<td>FES282rif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>29</td>
<td>FES292rif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>30</td>
<td>SWP302ria.ueim</td>
</tr>
<tr>
<td>FES</td>
<td>31</td>
<td>FES312rif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>32</td>
<td>SWP322ria.ueim</td>
</tr>
<tr>
<td>FES</td>
<td>33</td>
<td>FES332rif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>34</td>
<td>SWP342ria.ueim</td>
</tr>
<tr>
<td>SWP</td>
<td>35</td>
<td>SWP352ria.ueim</td>
</tr>
<tr>
<td>LWP</td>
<td>36</td>
<td>LWP362rib.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>37</td>
<td>SWP372ria.ueim</td>
</tr>
<tr>
<td>LWP</td>
<td>38</td>
<td>LWP382ria.ueim</td>
</tr>
<tr>
<td>SWP</td>
<td>39</td>
<td>SWP392ria.ueim</td>
</tr>
<tr>
<td>LWP</td>
<td>40</td>
<td>LWP402rib.dat</td>
</tr>
</tbody>
</table>

Figure 5-6: An instance of Rilog cluster
Define cluster IUEIM.Pilog (  
cam char[3];  
img int;  
imagename char[*];  
) store as Ingres(Pilog);

Figure 5-7: The unified schema of the Rilog cluster

The Rilog and Pilog tables are stored in the Ingres Database Management System at the IUEIM node of the National Space Science Data Center at the Greenbelt, Maryland.

The IUEIM node of the ADS network also stores all the images pointed to by the Pilog and Rilog clusters. The Pilog and Rilog clusters are inventories of the actual images stored at the IUEIM node. The node also maintains inventories of other image products such as line-by-line spectra, high dispersion spectra and lotus compatible images.
<table>
<thead>
<tr>
<th>Camera</th>
<th>img</th>
<th>Image_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>LWP</td>
<td>1</td>
<td>LWP11pia.ueiim</td>
</tr>
<tr>
<td>LWP</td>
<td>2</td>
<td>LWP21pia.ueiim</td>
</tr>
<tr>
<td>LWP</td>
<td>3</td>
<td>LWP32pib.ueiim</td>
</tr>
<tr>
<td>LWP</td>
<td>4</td>
<td>LWP41pia.ueiim</td>
</tr>
<tr>
<td>LWP</td>
<td>5</td>
<td>LWP51pia.ueiim</td>
</tr>
<tr>
<td>LWP</td>
<td>6</td>
<td>LWP62pia.ueiim</td>
</tr>
<tr>
<td>LWP</td>
<td>7</td>
<td>LWP71pia.ueiim</td>
</tr>
<tr>
<td>SWP</td>
<td>8</td>
<td>SWP81pib.ueiim</td>
</tr>
<tr>
<td>SWP</td>
<td>9</td>
<td>SWP91pia.ueiim</td>
</tr>
<tr>
<td>SWP</td>
<td>10</td>
<td>SWP102pib.ueiim</td>
</tr>
<tr>
<td>SWP</td>
<td>11</td>
<td>SWP111pib.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>12</td>
<td>SWP122pib.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>13</td>
<td>SWP131pia.dat</td>
</tr>
<tr>
<td>FES</td>
<td>14</td>
<td>FES142pif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>15</td>
<td>FES152pif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>16</td>
<td>FES161pia.dat</td>
</tr>
<tr>
<td>FES</td>
<td>17</td>
<td>FES172pif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>18</td>
<td>FES181pif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>19</td>
<td>FES192pif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>20</td>
<td>FES202pif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>21</td>
<td>SWP212pia.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>22</td>
<td>SWP221pia.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>23</td>
<td>SWP231pib.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>24</td>
<td>SWP241pib.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>25</td>
<td>SWP252pia.dat</td>
</tr>
<tr>
<td>FES</td>
<td>26</td>
<td>FES262pia.dat</td>
</tr>
<tr>
<td>FES</td>
<td>27</td>
<td>FES272pif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>28</td>
<td>FES282pif.dat</td>
</tr>
<tr>
<td>FES</td>
<td>29</td>
<td>FES292pif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>30</td>
<td>SWP302pia.dat</td>
</tr>
<tr>
<td>FES</td>
<td>31</td>
<td>FES312pif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>32</td>
<td>SWP322pia.dat</td>
</tr>
<tr>
<td>FES</td>
<td>33</td>
<td>FES332pif.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>34</td>
<td>SWP342pia.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>35</td>
<td>SWP352pia.dat</td>
</tr>
<tr>
<td>LWP</td>
<td>36</td>
<td>LWP362pib.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>37</td>
<td>SWP372pia.dat</td>
</tr>
<tr>
<td>LWP</td>
<td>38</td>
<td>LWP382pia.dat</td>
</tr>
<tr>
<td>SWP</td>
<td>39</td>
<td>SWP392pia.dat</td>
</tr>
<tr>
<td>LWP</td>
<td>40</td>
<td>LWP402pib.dat</td>
</tr>
</tbody>
</table>

Figure 5-8: An instance of Pilog cluster
5.1.2 Infrared Processing and Analysis Center (IPAC)

The Infrared Processing and Analysis Center, located at Jet Propulsion Laboratories in California, maintains observations made by the IRAS mission on a CDC/Cyber computer in IMDM database management system. The important data sets stored at this center include the point source catalog, the all sky survey, the additional observations and the faint source survey.

The Point Source Catalog (Iraspsc)

The point source catalog (Iraspsc) maintains the profile of about 250,000 well confirmed points of the infrared sources. The profile includes their position (ra and dec coordinates), the object id, the major axis, the position angle, the class, the wavelength, various flux densities observed and signal to noise ratio of each observation.

The point sources catalog at the Infrared Processing and Analysis Center (IPAC) is stored as a hierarchical database in the IMDM DBMS on a CDC/Cyber machine. The integrated schema of the hierarchical Iraspse is described in Figure 5-9. An instance of the Iraspse cluster is shown in Figure 5-10.
Define cluster IPAC.Irascpc (  
PSC table (  
   oid   char[10];  
   key   int;  
   ra    int;  
   dec   int;  
   major int;  
   posan int;  
   class int;  
   Observation table (  
   wav   int;  
   fl    int;  
   qfl   int;  
   snr   float;  
   ufl   int;  
   )  
  )  
)  
) store as IMDM(ipac_qcat);

Figure 5-9: The unified schema of the Irascpc cluster
<table>
<thead>
<tr>
<th>Objid</th>
<th>key</th>
<th>Ra</th>
<th>Dec</th>
<th>major</th>
<th>Posan</th>
<th>CI</th>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>wav fl qfl snr ufl</td>
</tr>
<tr>
<td>ori_nebu</td>
<td>i2</td>
<td>2988</td>
<td>-1950</td>
<td>180</td>
<td>129</td>
<td>72</td>
<td>15 18 17 .01 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20 23 22 .01 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25 42 39 .01 40</td>
</tr>
<tr>
<td>jupiter</td>
<td>i1</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>132</td>
<td>3</td>
<td>10 18 17 .02 20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15 23 20 .02 21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20 42 40 .03 37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25 50 49 .01 50</td>
</tr>
<tr>
<td>m42area</td>
<td>i3</td>
<td>2996</td>
<td>-1962</td>
<td>180</td>
<td>137</td>
<td>72</td>
<td>10 15 21 .01 22</td>
</tr>
<tr>
<td>moon</td>
<td>i4</td>
<td>0</td>
<td>0</td>
<td>90</td>
<td>163</td>
<td>2</td>
<td>10 15 23 25 .02 25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15 50 48 .04 47</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20 23 22 .03 21</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25 42 49 .05 47</td>
</tr>
<tr>
<td>nova_vul</td>
<td>i5</td>
<td>1097</td>
<td>7716</td>
<td>270</td>
<td>57</td>
<td>72</td>
<td>10 18 16 .01 16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15 23 21 .01 21</td>
</tr>
<tr>
<td>jupiter</td>
<td>i6</td>
<td>0</td>
<td>0</td>
<td>180</td>
<td>135</td>
<td>2</td>
<td>10 28 26 .01 26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25 42 41 .01 41</td>
</tr>
<tr>
<td>md11a</td>
<td>i7</td>
<td>3216</td>
<td>-2010</td>
<td>180</td>
<td>145</td>
<td>3</td>
<td>10 28 21 .04 22</td>
</tr>
</tbody>
</table>

Figure 5-10: An instance of Iraspse cluster.

5.1.3 Smithsonian Astrophysics Observatory (SAO)

The Smithsonian Astrophysics Observatory (SAO) maintains data that is gathered using the X-ray instruments. The observed images, spectra and other properties of the objects that are visible at the X-ray band are maintained at this center. One of the data sets which is a catalog of all the X-ray point sources is described subsequently.

The Sao Catalog

The Sao catalog maintains the profile of over 20,000 confirmed points of the X-ray sources. The profile includes the position defined by the ra (right ascension)
and dec (declination) coordinates, the camera name, the object id, the sequence
number, the class, the title, the time of observation, the exposure time, and the
instrument name.

The catalog of X-ray observations called Sao is stored at the Center for
Astrophysics at Harvard, Massachusetts. The catalog is stored in the DAVID DBMS
as a network cluster. The schema for the catalog is described in Figure 5-11. An
instance of the SAO cluster is shown in Figure 5-12.

```
Define cluster CFA.SAO (  
   Source table (  
      objid     char[10];  
      ra        int;  
      dec       int;  
      seq       int;  
      cl        int;  
      instrument table;  
   );  
   field table (  
      title     char[30];  
      stim      int;  
      etim      int;  
      cra       int;  
      cdec      int;  
      instrument table;  
   );  
) store as DAVID(qbase);  
define instrument table ( instr  char[3]);
```

Figure 5-11: The unified schema of the Sao cluster
<table>
<thead>
<tr>
<th>System</th>
<th>Field</th>
<th>Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>Title</td>
<td>stmin</td>
</tr>
<tr>
<td>1</td>
<td>I234</td>
<td>0930</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1 IPC</td>
</tr>
<tr>
<td>3</td>
<td>A786</td>
<td>1729</td>
</tr>
<tr>
<td>3</td>
<td>2 IPC</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5 Jupiter</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>6 hd2151</td>
<td>2930</td>
</tr>
</tbody>
</table>

Figure 5-12: An instance of Sao cluster.

5.1.4 A Science Scenario

A typical query used by NASA scientists for accessing the data from the Astrophysics Data System is presented here. In general, scientists are interested in obtaining the images or spectra of certain area or the objects in space that satisfy specific criteria. A sample query may be:

*Find the photometrically corrected images observed by the IUE mission for all the objects which have also been observed by the x-ray mission of Smithsonian Astrophysics Observatory as well as the IRAS mission. Also the IRAS observations about the objects were made at 10 micron wave length with flux density greater than 10.*

In GSQL the above query is expressed as follows:
SELECT t.ra, t.dec, t.objid, u.image_name
FROM  Iraspce r, Sao s,
       Iuelog t, Pilog u
WHERE r.flux > 10
    and r.wave = 10
    and r.objid = s.objid
    and s.objid = t.objid
    and t.cam = u.cam
    and t.img = u.img;

The query graph as defined in [51] for the above query is given in Figure 5-13.

![Figure 5-13: Query Graph for the GSQL Query](image)

This query uses the source clusters Iraspce stored at IPAC in CA, Sao stored at Center for Astrophysics in MA, Pilog is stored at the IUEIM node at Goddard Space Flight Center in MD and Iuelog stored at IUE-RDAF in MD. It requests the object id, the ra and dec coordinates and the image name of all the objects that satisfy the boolean condition.
5.2 Statistical Estimation

The query decomposition algorithm is dependent upon the technique it employs for the estimation of the sizes of intermediate results. The objective of an optimizer is to derive an efficient plan for the execution of a query. In order to accomplish this a query optimizer generates and evaluates a number of alternative strategies.

One of the critical inputs to the query decomposition process is the size estimation of all the intermediate clusters necessary to answer a query. The actual size of an intermediate result of any operation is known only after executing the operation. However, performing the actual operation to determine the size of its result cluster defeats the very purpose of the query decomposition process, which is to determine the best sequence of operations prior to their actual execution.

The problem of estimation can be expressed as: how can one evaluate the size of the result cluster of a cluster algebra operation without actually performing the operation.

Statistical information and techniques are the primary means for estimating the sizes of the result clusters. The common quantitative descriptors used by statisticians to summarize the data for the purpose of making inferences can be classified as follows:

1. Central tendencies such as mean, median and mode.

2. Dispersions such as variance, standard deviation, and range.
3. Size Parameters such as number of occurrences and number of distinct occurrences.

4. Parametric distributions such as the uniform, the normal and exponential distributions.

5. Non-parametric distributions such as value intervals and counts.

The statistical information about the clusters may consists of any one or combination of these descriptors. The choice of descriptors is governed by the accuracy of estimation, the time and the space considerations.

The conceptual clusters that are referred to in a query are called base clusters. The application of a cluster algebra operation on the base clusters produces an intermediate cluster. A query execution tree is a hierarchically structured tree [11]. The leaf nodes of such an execution tree are the base clusters. The internal nodes are intermediate clusters. Edges of the tree represent the cluster algebra operations. The stat-info for the base cluster is generated in advance and is updated periodically. The statistical information (stat-info) of the attributes of the intermediate clusters can not be computed because during the query decomposition process the operations of the execution tree are not performed. Thus, the stat-info of attributes of the intermediate clusters has to be estimated from the operations and the operand clusters. The stat-info of the attributes of the intermediate cluster is always estimated. As a result, there may be a certain degree of inaccuracy associated with the sizes of intermediate results and the stat-info.
In the following section, the type of statistical information along with the data structures used to gather this information are described. In section 5.2.1.3, the estimation methodology utilizing non-parametric statistical information for the selection ($\sigma$), normal projection ($\pi$), normal join (NJN) and extended semijoin ($\bowtie$) operations is presented. These techniques are illustrated using the sample database described in the previous section. In section 5.2.2, propositions concerning the accuracy of the proposed estimation procedures are discussed. Additionally, a comparison of this estimation method to the selectivity approach in the case of the join operation, assuming that the underlying DBMSs are relational, is described.

5.2.1 Estimation of the Intermediate Cluster Sizes

The quantitative descriptors used for the describing the statistical information employ the non-parametric distribution. The stat-info maintained for the base clusters consists of the attribute values and the number of cluster rows with each value. For the key fields the number of cluster rows is either zero or one. A separate data structure is employed to maintain stat-info of the key attributes. The method proposed here is practical for the databases that satisfy one of the following criteria: (1) either the database clusters are small in size or (2) the most frequently queried attributes have small number of unique values, irrespective of the actual number of rows in a cluster.

To illustrate the usefulness of the method, the examples of astrophysics, earth observations, and other NASA mission databases are used. The clusters shown in Figures 5-4, 5-6, 5-8 and 5-10 are a sample of databases of the astrophysics missions.
Most of the queries on these database clusters are expressed over the attributes such as the class, the wavelength (of observations), the flux density, the alignment of major/minor axis and the position angles.

Each mission is designed to observe a finite set of wavelengths accurately e.g. IRAS mission observed infrared sky at only five wavelengths. The whole spectrum of objects in the universe is divided into 99 classes (e.g. 1 implies extra-galactic object). Similarly, depending on each satellite mission there are a limited number of orientation angles for major, minor axis and positions.

The characteristics described above apply to many of the mission databases of NASA. The mission databases are a significant NASA information resource. Each mission database cluster has large number of rows (a few thousand to half a million) with a limited number of unique values for the most frequently queried attributes. It is very important to have best possible estimates since an error in the estimate may lead to an execution tree with significantly higher cost of evaluation.

5.2.1.1 The Statistical Information and Data Structures

The main data structures designed for the purpose are the stat-info files, directory files and several other data matrices. The data matrices contain the information about the clusters. The directory contains the cluster schema, the storage map of the cluster, the size of a cluster row (csz), the number of cluster rows (crows), information about the tables such as the size of a table row (tsz), the number of table rows (trows) and attributes of each table (aname), their types (type), sizes (asz) and
the location of the relevant stat-info (ptr_statinfo). An abridged version (relevant to the statistical estimation) of the directory is shown in Figure 5-14. The directory is stored as a hierarchical cluster.

![Figure 5-14: An Abridged Version of Directory](image)

The estimation process requires presence of certain statistical information about the attributes of a cluster. The stat-info for a cluster, C, consists of the following:

1. The size of a table row for each table in a cluster which is represented by $\text{Width}_j(C)$ denoting the size of a row in the $j^{th}$ table of cluster $C$.

2. The number of tuples in each table of a cluster which is represented by $\text{Tuple}_j(C)$ denoting the number of tuples in the $j^{th}$ table of cluster $C$.

3. The number of tables in the cluster $C$ is denoted by $\text{TABLES}[C]$. 
4. For all the important attributes, the number of occurrences of each attribute value in a cluster. This information is maintained as follows:

(A) The number of rows in a cluster that have the specific attribute value is maintained. For example, $A_c[v_k, C-row]$ denotes the number of cluster rows with value $v_k$ for the non-key attribute $A$ in cluster $C$. Also, $K^A_c[v_k]$ denotes the presence of the value $v_k$ for the key attribute $A$ in cluster $C$. The value is "1" if $v_k$ is present in a cluster and "0" otherwise.

(B) For each table in a cluster, the number of table rows that are associated with the specific attribute value is maintained. For example, $A_c[v_k, t-row_j]$ denotes number of tuples with value $v_k$ of attribute $A$ in the $j$th table of cluster $C$.

5. The association matrices maintain the information regarding the associations between the attribute values of one attribute to that of another attribute. $ASN_c^{AB}[v_i, v_j, C-row]$ Denotes number of cluster rows in cluster $C$ with attribute values $A = v_i$ and $B = v_j$.

The data elements that store the stat-info for an attribute of a cluster are described as follows:

```c
STRUCT stat-info {
    ATTR_TYPE attr_value;
    int c_row_count;
    int t1_row_count;
    int t2_row_count;
}
```
Where \texttt{attr\_value} is a value from the domain of the attribute that is present in a cluster. The \texttt{c\_row\_count} is number of the cluster rows with the attribute value in the cluster. The \texttt{ti\_row\_count} is a count of the \textit{i^{th}} table rows associated with the attribute value.

The total number of such data elements to be maintained are same as the number of unique values for the attribute in a cluster. The upper limit on the number of data elements is the cardinality of the domain of an attribute. If the number of unique values for an attribute cluster is not prohibitive then the information can be maintained as the linked list of the data elements. In the cases where the domains have large number of values the stat-info is organized as B\_tree to reduce the cost of accessing the relevant data.

The key attributes have only one occurrence in a cluster. Thus, for key attributes, there is no need to maintain a number of count fields. A simple data structure capable of recording the presence or absence of the key value is sufficient. An array of bits can be used to record such information. The domain of the attribute is mapped to an ordered integer value range. The range of a domain becomes 1,2,3,4, ... \(N\) where \(N\)
is a positive integer. The presence of an attribute value in the cluster is recorded by setting the bit value at the corresponding range position of the array to 1. The absence is recorded by zero.

The association matrices maintain information regarding the correspondence between the attribute values of the one attribute to that of another attribute.

The association information for two attributes A and B of a cluster C, denoted by \( \text{ASN}_{C}^{AB} \), consists of cluster rows for each unique combination of the A and B values. The combinations for which the count of cluster rows is zero are not stored in the matrix. The absence of any valid combination in the matrix implies zero count for that combination. The data element to store the information for any valid combination is described as follows:

```c
struct ASN_info {
    ATTR_TYPE1 A;
    ATTR_TYPE2 B;
    int    C_row;
};
```

The data element described above are stored as a linked list or B+ tree depending on the number of such elements. If the number is small then the former is the method of choice. For large number of the data elements a B+ tree is used to reduce the access time.

For the important attributes of the cluster instances shown in Figures 5-4 and 5-10, the stat-info and association information is given subsequently. The stat-info
for the attributes of the Iraspsc cluster is presented as a set of 4-tuples consisting of
the attribute value, the number of cluster rows with the specific attribute value, the
number of Psc table rows and the number of Observation table rows with the specific
attribute values. For example, the stat-info for the wave attribute is as follows:

\[
<10,5,5,5> <15,5,5,5> <20,3,3,3> <25,4,4,4>
\]

The stat-info for the class attribute is as follows:

\[
<2,6,2,6> <3,5,2,5> <72,6,3,6>
\]

The association information between the wave and class attributes is stored as
a set of 3-tuples consisting of the value of the wave attribute, the value of the class
attribute and the number of the cluster rows with both attribute values. For the Iraspsc
cluster instance of Figure 5-10 the association information is given as follows:

\[
<10,2,2> <15,2,1> <20,2,1> <25,2,2> <10,3,2> <15,3,1>
<20,3,1> <25,3,1> <10,72,1> <15,72,3> <20,72,1> <25,72,1>
\]

Additionally, the stat-info for the class attribute of the Iuelog cluster instance of
Figure 5-4 is represented by 2-tuples, consisting of the attribute value and the number
of cluster rows with the specific attribute value. For the cluster shown in Figure 5-4
the stat-info of the class attribute is as follows:

\[
<2,4> <3,4> <63,4> <72,13>
\]
5.2.1.2 Operators on the Stat-info

The stat-info is created, updated and used for estimating the size of the result of a cluster algebra operation. The estimation process also derives stat-info of the attributes of the result clusters for further estimations. The internal structure for maintaining the data elements may be a linked list, a B+_tree or any other efficient access structure. For the purpose of maintaining the data elements, the following abstract operations that hide the internal representation are implemented.

1. Create_stat-info (clustername, attribute_name) : This operation sets up the proper internal data structure (e.g. a linked list or a B+_tree). Also, it updates the ptr_statinfo field in the DIRECTORY for the given cluster name and attribute name.

2. Insert_stat-info (clustername, attributename, value, countlist) : This operation searches the stat-info file associated with the clustername and attributename for the attr_value = value. If the data element with attr_value = value is found, then all the count fields (e.g. c_row_count, t1_row_count etc) in the data element are added to by the values supplied in the countlist. Otherwise, a new data element is created and inserted into the data structure with count values set to the ones supplied by the countlist.

3. Delete_stat-info (clustername, attributename, value, countlist): This operation searches the stat-info file associated with the clustername and attributename. If the data element with attr_value = value is found then all count fields such as c_row_count, t1_row_count etc are decrement by the
numbers supplied in the countlist for the respective counts.

4. Get\_stat-info (clusternamex, attributename, value, countlist): This operation searches the stat-info file associated with the clusternamex and attributename. If the data element with attr\_value=value is found then all the countfield information is set to the countlist and returned. Otherwise, all the elements of the countlist are set to the zero value.

The stat-info is stored and manipulated using the above abstract data type operations. An optimizer assumes that all the stat-info about the relevant attributes is present and up-to-date and uses it for the estimation purposes. To accomplish this task the following operations are introduced.

1. Build\_stat-info (clusternamex, attributename): This operation builds the stat-info data structure for an attribute of a cluster specified as a parameter. The function uses create\_stat-info to create an appropriate data structure. The process reads one cluster row at a time from the named cluster and scans the cluster buffer to compute the number of table rows of each table associated with the current value of the attribute and stores in the countlist. Finally, it calls the insert\_stat-info function to store the information. The Figure 5-15 shows the data flow diagram for this operation.

2. Update\_stat-info (clusternamex, attribute, db\_operation): In general, the information stored in clusters is dynamic in nature. Over a period of time some rows are added and deleted from the cluster rendering outdated stat-info of attributes. For estimations to be accurate, the stat-info should always be
current. The update_stat-info function is used for dynamically updating the stat-info of the attributes in a cluster. The data flow diagram for such a function is shown in Figure 5-16. Any time a row in a cluster is inserted or deleted a call is made to the update_stat-info function. Inputs to this function are cluster name, attribute names and the database operation. Depending upon the database operation, this function internally calls insert_stat-info or the delete_stat-info functions.

3. Estimate_stat-info( cluster, attribute, db_operation) : This function estimates the intermediate result sizes and the distribution of attributes of the result clusters of the cluster algebra operations. The estimate function accepts cluster names, cluster algebra operation and the attribute name whose stat-info
is to be estimated as input and derives the stat-info of the attribute. The actual algorithm used to estimate the stat-info of an attribute of the output result cluster depends upon the cluster algebra operation and the type of the attribute being estimated. For each cluster algebra operation an specialized function is used to estimate the stat-info. These specialized functions are discussed in the following sections in detail. In this work, the attributes in a cluster algebra query are classified in two categories 1) The explicit condition attributes are attributes that appear in the condition part of the cluster algebra operator. e. g.

\[ \sigma_{A_{10}}(C_1) \]  \hspace{1cm} (1)

or

\[ C_1 \ X_{c_1, A_{10}=c_2} \ A \ C_2 \]  \hspace{1cm} (2)

In both of the above examples, A is an explicit condition attribute. 2) Implicit attributes are those attributes that appear in the result cluster and are not part of the condition in the current cluster algebra operation. The implicit attributes that are part of the condition in a subsequent cluster algebra operation in an execution tree are referred to as the implicit active attribute. On performing each cluster algebra operation in an execution tree the stat-info of all the explicit and implicit active attributes needs to be estimated. The size of result cluster can be computed by the information available in the stat-info of the explicit attribute.

In much of the literature, the estimation of statistical information of the implicit active attribute has been done under the independence of the attributes assumption.
An algorithm to derive the statistical information of implicit attributes with dependency is presented. The dependency between the attributes is stored as the association matrix.

![Dataflow Diagram for Estimate_stat-info](image)

Figure 5-17: Dataflow Diagram for Estimate_stat-info

### 5.2.1.3 Estimation Methodology

The estimation methods used for the select, normal project, normal join, semijoin cluster algebra operators are discussed in the following sections. The estimation procedures derive the stat-info of the attributes in the result cluster of the cluster algebra operation without actually executing the operation. The approach presented here is valid for both the base and the intermediate clusters of an execution tree.

The size of the result of an operation can be computed by using the derived stat-info of an attribute in the result cluster. The size of a cluster \( C \) is actually the sum of the sizes of all the tables in that cluster. The size of the \( i^{th} \) table can be computed by the product of the number of tuples in the \( i^{th} \) table (tuple\(_i[C]\)) and the size of each tuple in bytes i.e., Width\(_i[C]\). Since TABLE\(_i[C]\) is the number of tables in a cluster \( C \), the size of \( C \) is computed as follows:

\[
SZ[C] = \sum_{i=1}^{\text{TABLE}[C]} (\text{Width}_i[C] \times \text{tuple sub } i[C])
\]  

(3)
The approach used to derive the stat-info of the explicit condition attributes for each cluster algebra operator is presented in Sections 5.2.1.3.4 through 5.2.1.3.4. Whereas, the method for estimating the stat-info for implicit active attributes is presented in Section 5.2.1.4.

5.2.1.3.1 Select

The select (σ) operator generates a horizontal subset of the operand cluster. All the cluster rows in the operand cluster that satisfy the selection condition appear in the output cluster. The selection condition is made up of a boolean combination of terms. Each term is a simple comparison that evaluates a condition as being either true or false. The terms are formed using logical operators (lop) such as <, > and =. An example of a term is A > 10. An example of the select operator on the Iraspsc cluster is given as follows:

$$σ_{wave ≥ 20} \ (Iraspsc)$$  \hspace{1cm} (4)$$

However, the selection condition may involve more than one term such as,

$$σ_{wave ≥ 20 \times flux > 10} \ (Iraspsc).$$

These compound conditions can be expressed as a sequence of simple conditions with only one term using the associative property of the select operator. The equivalent selection expression for the above expression is

$$σ_{wave ≥ 20} (σ_{flux > 10} \ (Iraspsc)) .$$

Thus, for the estimation process each select operator is presumed to have a simple condition of the form $σ_{Attribute < lop > Constant (C)}$. 

The stat-info of the explicit condition attributes of the output cluster can be derived from the stat-info of the attribute in the operand cluster. Let $\sigma_{A < \text{lop} > S}$ (C) be the select operation. The stat-info of the attribute A in the output cluster (CR) is estimated as follows:

Case I. If A is a non-key attribute in cluster C, then:

$$\forall v_j \in \text{Dom}(A) \quad A_{C^R}(v_j) = \begin{cases} A_c(v_j) & \text{if } v_j < \text{lop} > S \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

(5)

For each table i in cluster C, the number of table rows after the selection operation on attribute A is as follows:

$$\forall v_j \in \text{Dom}(A) \quad A_{C^R}(v_j) = \begin{cases} A_c(v_j, T_{row_i}) & \text{if } v_j < \text{lop} > S \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

(6)

Case II. If A is a key attribute in cluster C, then:

$$\forall v_j \in \text{Dom}(A) \quad K^A_{C^R}(v_j) = \begin{cases} K^A_c(v_j) & \text{if } v_j < \text{lop} > S \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

(7)

For example, the stat-info of the wave attribute of cluster CR where $C^R = \sigma_{\text{wave} \geq 20} (\text{Iraspse})$ is as follows:

$<20,3,3,3> \quad <25,4,4,4>$
5.2.1.3.2 Normal Project

The operation of the normal projection generates a vertical subset of a cluster. The output cluster is a relational table. The subset is obtained by picking the specified attributes from the operand cluster and eliminating others.

Let $\pi_{A, B, D, \ldots} (C)$ be the projection operation and let $C^R$ be the output cluster of this operation. The stat-info of attribute $A$ of the cluster $C^R$ is derived as follows:

Case I. Let $L = \{A, B, D, \ldots\}$ be the list of all the attributes projected by the operation. Let the list $L$ be a subset of the attributes of a table $(i)$ of the cluster $C$, then

$$\forall v_k \in \text{Dom}(A) \quad A_{C^R}(v_k, C_{row}) = A_C(v_k, T_{row_i}) \quad (8)$$

Case II. List $L$ consists of the attributes from more than one table of the cluster.

$$\forall v_k \in \text{Dom}(A) \quad A_{C^R}(v_k, C_{row}) = A_C(v_k, C_{row}) \quad (9)$$

In this case, all the information regarding the number of table rows that are associated with the specific values of attribute $A$ are eliminated from the stat-info for $A$ in $C^R$. The result cluster consists of one table.

The stat-info of the class attribute of cluster $C^R$ where $C^R = \pi_{\text{class, objid, img}}(\text{filelog})$ is as follows:

$$<2,4> <<3,4> <63,4> <72,13>$$
5.2.1.3.3 Normal Join

The normal join is a binary operation. It takes two clusters as its operands. If the joining attributes of the two clusters have a common domain, the clusters are joined to produce a result cluster. The result cluster consists of the attributes of both clusters and is a relational table. Each row of the result cluster consists of the values from the cluster rows of the two operand clusters that agree on the joining attribute values.

A method to estimate the stat-info, of the explicit condition attribute of the output cluster is presented here. Let C1 and C2 be the cluster being joined (C1 NJN C1.A = C2.A C2) and CR be the result of the normal join operation. The result CR will be in normal form. All the attributes of this cluster have atomic domains. The derived stat-info of the joining attribute in the result cluster is 2-tuple consisting of the attribute value and the number of cluster rows with the specific attribute value. The estimation method depends on the type of attributes involved in the joining condition. For estimation purposes, the join operations that involve different combinations of key and non-key attributes are treated differently.

Case I. If C1.A and C2.A are non-key attributes, then the stat-info of A in the result CR can be derived as follows:

\[
\forall x \in \text{dom}(A) \quad A_{C^R}(v_k, C_{-row}) = A_{C1}(v_k, C_{-row}) * A_{C2}(v_k, C_{-row})
\]

(10)
Case II. Let C1.A be the key attribute while C2.A be the non-key attribute. The stat-info of the attribute A in C^R is derived by:

\[ \forall v_k \in \text{dom}(A) \quad A^A_{C^R}(v_k, C \_ \text{row}) = K^A_{C1}(v_k) \star A_{C2}(v_k, C \_ \text{row}) \quad (11) \]

Case III. If C1.A and C2.A are key attributes, then attribute A in C^R will also be the key of cluster C^R. The stat-info of attribute A in C^R is derived as follows:

\[ \forall v_k \in \text{dom}(A) \quad K^A_{C^R}(v_k) = K^A_{C1}(v_k) \star K^A_{C2}(v_k) \quad (12) \]

The stat-info of the class attribute in the result of the normal join of Iuelog and Iraspnc cluster instances over the class attribute, i.e. (Iuelog) NJN\text{\text{class}} (Iraspnc), is as follows:

\( <2,24> <3,20> <72,78> \)

5.2.1.3.4 Extended Semijoin

The semijoin operation is equivalent to joining two clusters and then projecting the result of the join over the schema of one of the clusters. The result of a semijoin, unlike that of a normal join, may be a cluster with embedded tables. In fact, it has the same schema as one of its operand clusters [2]. In this work, it is always the schema of the first cluster. The semijoin reduces the size of its first cluster by eliminating those cluster rows that will not satisfy the joining condition. A semijoin is expressed as \( C1 \alpha_{C1.A = C2.A} C2 \). The cluster C1 is reduced by this operation.
Let $F(x)$ be a function which returns value 1 whenever $x > 0$ and returns value 0 otherwise. i.e.,

$$F(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$$ (13)

Case I. If C1.A and C2.A are non-key attributes, then the stat-info of A in the result $C^R$ is derived as follows:

$$\forall v_k \in \text{dom}(A) \quad A_{c^r}(v_k, C\_row) = A_{c1}(v_k, C\_row) * F(A_{c2}(v_k, C\_row))$$ (14)

For Each Table i in cluster C1,

$$\forall v_k \in \text{dom}(A) \quad A_{c^r}(v_k, T\_row_i) = A_{c1}(v_k, T\_row_i) * F(A_{c2}(v_k, C\_row))$$ (15)

Case II. Let C1.A be the key attribute while C2.A be the non-key attribute. Then, the stat-info of the attribute A in $C^R$ is derived as follows:

$$\forall v_k \in \text{dom}(A) \quad K_{c^r}(v_k) = K_{c1}(v_k) * F(A_{c2}(v_k, C\_row))$$ (16)

Case III. The attribute C2.A is a key attribute while C1.A is a non-key attribute. The stat-info of A in the result cluster $C^R$ is derived as follows:

$$\forall v_k \in \text{dom}(A) \quad A_{c^r}(v_k, C\_row) = A_{c1}(v_k, C\_row) * K_{c2}(v_k)$$ (17)
For Each Table \( i \) in cluster \( C_1 \),

\[
\forall_{v_k \in \text{dom}(A)} A_{c,k}(u_k, T_{-row_i}) = A_{c1}(u_k, T_{-row_i}) * F(A_{c,k}(u_k, C_{-row}))
\]  

(18)

Case IV. If \( C_1.A \) and \( C_2.A \) are key attributes, then attribute \( A \) of \( C^R \) will also be a key attribute. The stat-info of \( A \) in \( C^R \) is derived as follows:

\[
\forall_{v_k \in \text{dom}(A)} K^{A}_{c,k}(u_k) = K^{A}_{c1}(u_k) * K^{A}_{c2}(u_k)
\]  

(19)

The stat-info of the class attribute of the result of the semijoin reduction of the Iuelog cluster instance by the Iraspse cluster instance, i.e. \( \text{Iuelog} \alpha_{\text{class}} \text{Iraspse} \) is as follows:

\(<2,4> <3,4> <72,13>\)

5.2.1.3.5 Method for the Estimation of the Implicit Active Attributes

Whenever a cluster algebra operation is performed on a cluster, the stat-info of all the attributes is affected. Thus, the stat-info of the implicit active attributes is estimated for use in later operations of the execution tree, where these attributes surface as explicit condition attributes.

The selection, normal join and semijoin operations modify the number of cluster rows of the result cluster thus altering the stat-info of the implicit active attributes. The method discussed here holds for all of the above cluster algebra operations. It uses the derived stat-info of an explicit condition attribute in the result cluster and
the association matrix between the two attributes of the input clusters. The method also derives a new association matrix between these two attributes for the result cluster.

Let A be an explicit condition attribute of the cluster C. Let \( C^R \) be the result cluster of a cluster algebra operation on cluster C. Let \( ASN_C^{AB} \) be the association matrix of attributes A and B in the cluster C. Then the stat-info of attribute B in the cluster \( C^R \) can be derived by following process:

1. Initialize:
   \[
   \forall v_j \in dom(B) \quad B_{c^k}(v_j, C_{row}) = 0 \quad (20)
   \]

2. For all \( v_j \) in Dom(B)
   for all \( v_k \) such that \( A_{c^k}(v_j, C_{row}) \neq 0 \)
   \[
   B_{c^k}(v_j, C_{row}) = B_{c^k}(v_j, C_{row}) + ASN_C^{AB}(v_k, v_j, C_{row}) \quad (21)
   \]
   \[
   ASN_C^{AB}(v_j, v_k, C_{row}) = ASN_C^{AB}(v_j, v_k, C_{row}) \quad (22)
   \]

The end result of above algorithm is a new association matrix of the attributes A and B in cluster \( C^R \) and the stat-info of B in cluster \( C^R \).

Let \( (\sigma \text{wave} \geq 20(\text{Iraspse}) \quad NJN_{class} \quad \text{fuelog}) \) be a cluster algebra query. In order to estimate the result size of this query, the following steps are required: (a) estimate the stat-info of the wave attribute, an explicit condition attribute, of the result cluster of the selection operation; (b) estimate the stat-info of
class, an implicit active attribute, of the result cluster of the selection operation; (c) estimate the stat-info of class, an explicit condition attribute, of the result cluster of the join operation between the temporary result of the selection operation and the fulelog clusters; (d) compute the size of the result cluster from the join operation.

The stat-info for the wave attribute in the result cluster of the selection operation is as follows:

\[<20,3,3> <25,4,4,4>\]

The stat-info and the ASN of the class attribute is as follows:

Stat-info: \[<2,3> <3,2> <72,2>\]

ASN: \[<20,2,1> <20,3,1> <20,72,1> <25,2,2> <25,3,1> <25,72,1>\]

The stat-info of the class attribute of the result cluster of the join operation is as follows:

\[<2,12> <3,8> <72,26>\]

Given that the \(\text{Width}(C^R) = 117\), the size of the result cluster is computed as follows:

\[\text{SZ}(C^R) = (12 + 8 + 26) \times 117 = 5382\]

5.2.2 Analysis of the Proposed Estimation Methods

In this section, the proposed method of estimation is evaluated. The results estimated by the proposed method are analyzed with reference to the results obtained
after the actual execution of the operation. The estimation method is also compared to the commonly used selectivity approach [7,13,62,64,65] that assumes the uniform distribution. The comparison is illustrated by executing the normal join operation over a relational cluster.

For the purpose of analysis the following notation is used:

Let A be an attribute of the cluster C_i. Dom(A) denotes the domain from which attribute A derives its values.

\[ N_i^l : \] denotes actual number of cluster rows in the cluster C_i
with the value \( v_j \in Dom(A) \)

\[ N_i^* = \sum_{v_j \in Dom(A)} N_i^l : \] denotes the total number of cluster rows in cluster C_i

\[ E_{ae}(alg\_exp) : \] The actual number of cluster rows obtained by the execution of the cluster algebra expression alg_exp.

\[ E_{sl}(alg\_exp) : \] The estimated number of cluster rows for the cluster algebra expression alg_exp using the estimation method described above. The method is also referred to as stat-info method.
5.2.2.1 Proposition 4.1

The number of cluster rows estimated by the stat-info method are the same as those obtained by the actual execution of the normal join operation.

Let C1 and C2 be two clusters with a common attribute A. Let \( E_{ae} \) be the total number of cluster rows in the cluster resulting from the actual execution of the normal join of C1 and C2.

Then the \( E_{ae} \) is given by:

\[
E_{ae} = \sum_{v_k \in \text{dom}(A)} N_k^1 \times N_k^2
\]

where, \( N_k^i \) denotes the actual number of cluster rows with value \( v_k \) in cluster Ci.

Let \( E_{si} \) be the total number of estimated cluster rows using the stat-info method. The computation of \( E_{si} \) depends on whether attribute A is a key or a non-key attribute of C1 and C2.

Case I. A is a key attribute in both the clusters C1 and C2 then:

\[
E_{si} = \sum_{v_k \in \text{dom}(A)} K_{C1}^A(v_k) \times K_{C2}^A(v_k)
\]
Where, \( K_{C_i}^A(v_k) \) has a value '1' or '0' depending upon the existence of the value \( v_k \) in the cluster \( C_i \). Since \( A \) is the key attribute, it can occur only in one cluster row of \( C_i \). This is the same as the total number of cluster rows of \( C_i \) with value \( v_k \) for the attribute \( A \). This is the same as \( N_i^k \).

Hence, we can rewrite,

\[
E_{si} = \bigvee_{v_k} \sum_{\epsilon \in \text{dom}(A)} N_1^k \cdot N_2^k
\]  

(25)

thus, \( E_{si} = E_{ae} \)  

(26)

Case II. Attribute \( A \) in a non-key attribute in clusters \( C_1 \) and \( C_2 \) then:

\[
E_{si} = \sum_{v_k \epsilon \text{dom}(A)} A_{C_1}(v_k, C_{row}) \cdot A_{C_2}(v_k, C_{row})
\]  

(27)

Where, \( A_{C_i}(v_k, C_{row}) \), is by definition the total number of cluster rows in the cluster \( C_i \) with value \( v_k \) for attribute \( A \). This is the same as \( N_i^k \) by definition.

Hence, \( E_{si} \) can be rewritten as:

\[
E_{si} = \bigvee_{v_k} \sum_{\epsilon \in \text{dom}(A)} N_1^k \cdot N_2^k
\]  

(28)
thus, \( E_{st} = E_{ae} \)  \hspace{1cm} (29)

Case III. Let A be key attribute in cluster C1 and a non-key attribute in cluster C2 then:

\[
E_{st} = \sum_{v_k \in dom(A)} K_{C1}^A(v_k) \ast A_{C2}(v_k, C_{row})
\]  \hspace{1cm} (30)

By the definition stated above, \( E_{st} \) can be rewritten as:

\[
E_{st} = \sum_{v_k \in dom(A)} N_1^k \ast N_2^k
\]  \hspace{1cm} (31)

\[
\text{thus, } E_{st} = E_{ae}
\]  \hspace{1cm} (32)

Thus, the stat-info estimation method computes the same results as the actual execution of the normal join operation.

5.2.2.2 Proposition 4.2

The number of cluster rows estimated by the stat-info method are same as those obtained by the actual execution of the selection operation.

Let A be an attribute, of the cluster \( C_i \) with domain as \( \text{Dom}(A) \). Let K be a value in the \( \text{Dom}(A) \). The expression \( A \ <lop> \ K \) is a condition where \( <lop> \in \{<, >, =, <>, \leq, \geq\} \).
Let $E_{ae}$ be the total number of cluster rows in the result of the selection operation

$$\sigma_{A <lop> \ k} (C_i)$$

after the actual execution of the operation then, the $E_{ae}$ is given as follows:

$$E_{ae} = \forall \ v_k <lop> \ k \ N_i^k$$  \hspace{1cm} (33)\]

The estimated number of cluster rows in the result of the above operation using stat-info method will depend upon the status of attribute $A$ in cluster $C_i$.

Case I. If $A$ is a non-key attribute of the cluster $C_i$ then,

$$E_{si} = \forall \ v_k <lop> \ k \ A_{C_i} (V_k, C_{row})$$  \hspace{1cm} (34)\]

Where, $A_{C_i} (V_k, C_{row})$ is by definition the total number of cluster rows in the cluster $C_i$ with value $v_k$ for attribute $A$. This is the same as $N_i^k$ by definition.

Hence, $E_{si}$ can be rewritten as:

$$E_{si} = \forall \ v_k <lop> \ k \ N_i^k$$  \hspace{1cm} (35)\]

thus, $E_{si} = E_{ae}$ \hspace{1cm} (36)\]

Case II. If $A$ is a key attribute of the cluster $C_i$ then,
\[ E_{si} = \sum_{v_k} \sum_{\langle \text{top} \rangle} K_{\hat{C}_i}^{A}(v_k) \]  

(37)

Where, \( K_{\hat{C}_i}^{A}(v_k) \) has a value '1' or '0' depending upon the existence of the value \( v_k \) in the cluster \( C_i \). Since \( A \) is the key attribute, it can occur only in one cluster row of \( C_i \). This is the same as the total number of cluster rows of \( C_i \) with value \( v_k \) for the attribute \( A \). This is the same as \( N_i^k \).

Hence, we can rewrite,

\[ E_{si} = \sum_{v_k} \sum_{\langle \text{top} \rangle} N_i^k \]  

(38)

thus, \( E_{si} = E_{ae} \)  

(39)

5.2.2.3 Proposition 4.3

The number of cluster rows estimated by the stat-info method are the same as those obtained by the actual execution of the semijoin operation.

Let \( C_1 \) and \( C_2 \) be two clusters with a common attribute \( A \). Let \( E_{ae} \) be the total number of cluster rows in the cluster resulting from the actual execution of the semijoin of \( C_1 \) and \( C_2 \).

Then the \( E_{ae} \) is given by:

\[ E_{ae} = \sum_{v_k} \sum_{\epsilon \in \text{dom}(A)} N_i^k * FACT \]  

(40)
Where

\[
F_{ACT} = \begin{cases} 
1 & N^k_2 > 0 \\
0 & \text{otherwise} 
\end{cases}
\]  
(41)

Let \( E_{si} \) be the total number of estimated cluster rows using the stat-info method. The computation of \( E_{si} \) depends on whether attribute A is a key or a non-key attribute of C1 and C2.

Case I. If A is a non-key attribute in both of the clusters \( C_1 \) and \( C_2 \) then, following the equation 13

\[
E_{si} = \sum_{\forall v_k \in \text{Dom}(A)} A_{C_i}(v_k, C_{-row}) \times F(A_{C_2}(v_k, C_{-row}))
\]  
(42)

Where, \( A_{C_i}(v_k, C_{-row}) \), is by definition the total number of cluster rows in the cluster \( C_i \) with value \( v_k \) for attribute A. This is the same as \( N_i^k \) by definition.

Hence, \( E_{si} \) can be rewritten as:

\[
E_{si} = \sum_{\forall v_k \in \text{Dom}(A)} N_1^k \times F(N_2^k)
\]  
(43)

Where \( F(X) \) is defined in the equation 13, hence

\[
F(N_2^k) = \begin{cases} 
1 & N_2^k > 0 \\
0 & \text{otherwise} 
\end{cases}
\]  
(44)

This is same as FACT defined above. Thus we can derive,
\[ E_{si} = \sum_{v_k \in \text{Dom}(A)} N_i^k \cdot F(A_{C_1}(u_k, C_{\text{row}})) \] (45)

thus, \[ E_{si} = E_{ae} \] (46)

Case II. If A is key attribute in cluster C_1 and non-key attribute in cluster C_2 then,

\[ E_{si} = \sum_{v_k \in \text{Dom}(A)} N_i^k \cdot F(A_{C_2}(u_k, C_{\text{row}})) \] (47)

Where, \( A_{C_i}(u_k, C_{\text{row}}) \), is by definition the total number of cluster rows in the cluster Ci with value \( v_k \) for attribute A. This is the same as \( N_i^k \) by definition.

Where, \( K_{C_i}(u_k) \) has a value '1' or '0' depending upon the existence of the value \( v_k \) in the cluster Ci. Since A is the key attribute, it can occur only in one cluster row of Ci. This is the same as the total number of cluster rows of Ci with value \( v_k \) for the attribute A. This is the same as \( N_i^k \).

Hence \( E_{si} \) can be rewritten as:

\[ E_{si} = \sum_{v_k \in \text{Dom}(A)} N_i^k \cdot F(N_2^k) \] (48)

Where \( F(X) \) is defined in the equation 13, hence

\[ F(N_2^k) = \begin{cases} 1 & N_2^k > 0 \\ 0 & Otherwise \end{cases} \] (49)
This is same as FACT defined above. Thus we can derive,

\[ E_{si} = \sum_{\forall v_k \in \text{Dom}(A)} N_1^k \cdot \text{FACT} \quad (50) \]

thus, \( E_{si} = E_{ae} \quad (51) \).

Case III. If A is a non-key attribute in cluster \( C_1 \) and key attribute in cluster \( C_2 \). Following the equation 13 the estimated number of cluster rows in the result will be:

\[ E_{si} = \sum_{\forall v_k \in \text{Dom}(A)} A_{c_1}(v_k, C_{\text{row}}) \cdot K^A_{c_2}(v_k) \quad (52) \]

Where, \( A_{c_1}(v_k, C_{\text{row}}) \), is by definition the total number of cluster rows in the cluster \( C_i \) with value \( v_k \) for attribute A. This is the same as \( N_1^k \) by definition.

\[ E_{si} = \sum_{\forall v_k \in \text{Dom}(A)} N_1^k \cdot K^A_{c_2}(v_k) \quad (53) \]

\[ K^A_{c_2}(v_k) = \begin{cases} 1 & \text{rows}(v_k) = 1 \\ 0 & \text{otherwise} \end{cases} \quad (54) \]

Since A is a key attribute in cluster \( C_2 \) the value \( v_k \) can occur only once in the cluster. Without loss of accuracy for a key attribute we can rewrite the above equation as following:
\[ K_{C_2}^A(v_k) = \begin{cases} 1 & \text{rows}(v_k) > 0 \\ 0 & \text{otherwise} \end{cases} \] (55)

The right hand side of above equation is same as of the one defining FACT. Hence, in this situation \( K_{C_2}^A(v_k) = \text{FACT} \). Thus, the \( E_{si} \) can be rewritten as:

\[ E_{si} = \sum_{\forall v_k \in \text{Dom}(A)} N_1^k \ast \text{FACT} \] (56)

thus, \( E_{si} = E_{\alpha_e} \) (57)

Case IV. If \( A \) is a key attribute in both of the clusters \( C_1 \) and \( C_2 \), then following the equation 13,

\[ E_{si} = \sum_{\forall v_k \in \text{Dom}(A)} K_{C_1}^A(v_k) \ast K_{C_2}^A(v_k) \] (58)

Where, \( K_{C_i}^A(v_k) \) has a value '1' or '0' depending upon the existence of the value \( v_k \) in the cluster \( C_i \). Since \( A \) is the key attribute, it can occur only in one cluster row of \( C_i \). This is the same as the total number of cluster rows of \( C_i \) with value \( v_k \) for the attribute \( A \). This is the same as \( N_i^k \).

Hence, we can rewrite,

\[ E_{si} = \sum_{\forall v_k \in \text{Dom}(A)} N_1^k \ast K_{C_2}^A(v_k) \] (59)
\[ K_{c_2}(v_k) = \begin{cases} 
1 & \text{rows}(v_k) = 1 \\
0 & \text{otherwise} 
\end{cases} \quad (60) \]

Since A is a key attribute in cluster C_2 the value v_k can occur only once in the cluster. Without loss of accuracy for a key attribute we can rewrite the above equation to

\[ K_{c_2}^A(v_k) = \begin{cases} 
1 & \text{rows}(v_k) > 0 \\
0 & \text{otherwise} 
\end{cases} \quad (61) \]

The right hand side of above equation is the same as that of the one defining FACT. Hence, in this situation \( K_{c_2}^A(v_k) = \text{FACT} \). Thus, the \( E_{si} \) can be rewritten as follows:

\[ E_{si} = \sum_{v_k \in \text{Dom}(A) \text{FACT}} N_k \quad (62) \]

thus, \( E_{si} = E_{a_0} \quad (63) \)

5.2.2.4 Comparison with Conventional Methods

The estimation methods based on the selectivity approach assuming the uniform distribution have been used to estimate the sizes of intermediate results [7]. Selectivity is defined as the number of the distinct attribute values occurring in a relation divided by all possible values for that attribute.
Let $C_1 \Join \frac{N_{1 \cdot A} \cdot N_{2 \cdot A}}{S}$ be a normal join operation. Let $S = |\text{domain}(A)|$

and $N_i^*$ be the total number of rows in cluster $C_i$. The estimated number of rows in the result, $N_{uni}$, using the selectivity approach assuming the uniform distribution is given as follows.

$$N_{uni} = \sum \left( \frac{N_1^*}{S} \cdot \frac{N_2^*}{S} \right)$$ (64)

The estimated number of rows using the stat-info method for the normal join operation is the same as one would obtain after the actual execution of the operation as shown above. Thus,

$$N_{si} = \vee_{u_k \in \text{dom}(A)} N_1^{k*} N_2^{k*}$$ (65)

Thus, the comparison between the selectivity and stat-info methods is equivalent to comparing the results obtained using the selectivity approach to the results obtained by actual execution of the operation, i.e.,

$$N_{si} - N_{uni} = \sum_{u_k \in \text{dom}(A)} N_1^{k*} N_2^{k*} - \frac{N_1^{*} \cdot N_2^{*}}{S}$$ (66)

The two techniques will estimate the same number of rows of the result of a join operation provided that $S^* \vee_{u_k \in \text{dom}(A)} N_1^{k*} N_2^{k*}$ is the same as $N_1^{*} \cdot N_2^{*}$.

Thus, in general, the estimates produced by the selectivity method assuming the
uniform distribution are different from the estimates produced by the stat-info method. It has been shown that the number of rows estimated by the stat-info method is the same as the actual number of rows obtained by executing the join operation. Hence, the stat-info method produces better results than the selectivity approach.

The estimation methodology proposed here is used for estimating the temporary cluster sizes for the query processing algorithm proposed and described in the subsequent section.

5.3 Query Processing

The existing distributed query processing algorithms proposed for the heterogeneous distributed database management systems has been discussed in the related research section. The query processing algorithms for the ADDS [66], IMDAS [63], MRDSM [67] decompose queries into a set of sub-queries that can be processed by a single database management system and sub-queries to assemble the result. The algorithm for Multibase [62], Mermaid [65], PRECI [64] decompose a query into a set of sub-queries consisting of operations for initial local processing, semijoins to reduce the communications cost, followed by the operations for assembling all the intermediate results at a single site to generate the final answer. These prototypes use semi-join based [7] algorithm and employ the selectivity [3,7] approach for the estimation purposes. The query processing algorithm for Mermaid also supports replicated data. It assumes all the component data is stored in the relational data model.
The query processing algorithm proposed here, decomposes a query expressed over the heterogeneous databases into a set of the cluster algebra operations. The sequence of operations generated for a query is also referred to as the execution tree. The execution tree consists of the basic cluster algebra operations such as the selection, the normal projection, the embed, the unembed, the normal join and the extended semijoin. The query decomposition algorithm generates the execution tree without actually performing any of the cluster algebra operations.

The query processing algorithm proposed and developed as a part of this work differs from the other query processing algorithms on the following points:

1. The proposed algorithm decomposes and generates execution trees for the queries expressed over the heterogeneous databases. Unlike the algorithms developed for the Mermaid, PRECI and IMDAS heterogeneous distributed database management systems that process queries over the heterogeneous databases consisting of the data stored in the relational model only, the proposed algorithm processes queries over the heterogeneous databases consisting of the data stored in the relational, hierarchical and the network data models.

2. The algorithm uses the stat-info method for the estimation of the intermediate result sizes. The technique produces accurate sizes for the cluster algebra operations without actually carrying out the operations. Also, the stat-info method takes into account the inter-dependence of attributes within a cluster.
3. It uses a combination of the normal join and the extended semijoin operations to generate the execution tree. The best sequence of normal join operations is produced using the zero-one integer linear programming methodology.

The principal modeling assumptions used in this research are: 1) The heterogeneous databases consist of the relational, hierarchical and network data models. 2) The cost of sending the message or the data is proportional to the number of bytes being transmitted over the network. The actual cost of sending includes the cost of the setup time plus the cost of sending the data. 3) The local processing cost is included in the algorithm. 4) The local access operations are grouped together whenever possible e.g. it may be possible to perform the selection and projection operations on the same cluster at the same time. 5) The materialization is given for a query. By materialization, one defines a non-redundant copy of entire distributed database upon which query is executed. 6) The estimated intermediate cluster sizes are derived using the stat-info method described in the preceding chapter.

The extended semi-join operation is an operation that accepts clusters instead of relations as its operands. The extended semi-join operation for clusters is a generalization of the semi-join operation for relations [7].

Let $c[S]$ and $d[S']$ be two clusters with schema $S$ and $S'$ respectively. The extended semi-join operation in cluster algebra, denoted by $c[S] \bowtie_A d[S']$ is equivalent to joining the clusters $c$ and $d$ over the common attribute $A$ and then projecting the result over
the schema \( S \). The cluster \( d \) in this case is referred to as the sending cluster and \( c \) as the receiving cluster. The result of the semi-join operation \( r \) is a subset of the original cluster. Thus, the \( r[S] \) is defined as follows:

\[
\pi_S(c \quad NJN \quad d)
\]

The operation of an extended semi-join in a distributed environment is performed by projecting out the attribute \( A \) from the sending cluster. The file containing the \( A \)-values of the cluster \( d \) is sent to the site of the receiving cluster. The value of the attribute \( A \) in the cluster \( c \) for each cluster row is matched against the \( A \)-values in the file. The cluster row is retained in the result, if a match is found. It is obvious from the process described that the result can not have anymore rows than the one present in \( c \). Thus, the result cluster of the semi-join operation is a subset of the receiving cluster. Further, the semi-join operator has the property of monotonically reducing the size of the result cluster.

In order to perform the extended semi-join operation the HDDBMS incurs in certain overhead costs. These overhead costs include the cost of transferring \( A \)-values of the cluster \( d \) to the site of the cluster \( c \) and the local processing cost of performing the reduction. The sum of the above two costs is referred to as the cost of the extended semi-join.

The extended semi-join operation as stated earlier reduces the size of the receiving cluster. The amount by which the size of the receiving cluster is reduced is referred to as the benefit of the extended semi-join. The profit of a semi-join is defined by the difference between the benefit and the cost of an extended semi-join.
For the purpose of presenting the query processing algorithm, it is assumed that the distributed heterogeneous network consists of M sites and there are N permanent clusters in the database. The following notation is used to denote the clusters and their sizes.

\( C_i^m[S_i] \): denotes a permanent cluster located at site m. The subscript \( i = 1, 2, ..., N \) denotes the \( i \)th cluster whereas the superscript \( m = 1, 2, ..., M \) denotes the site at which this cluster is located and \( S_i \) denotes the schema of the cluster.

\( S \in (C_i^m) \): denotes the size of the cluster \( C_i^m \) in bytes.

As a result of the cluster algebra operations, new temporary clusters are created. The result clusters of the cluster algebra operations such as select and semijoin have the same schema as that of original cluster. These clusters are truly horizontal subsets of the original clusters and are referred to as versions of the original cluster.

\( C_i^{(n)}[S_i] \): denotes a \( n \)th version of a permanent cluster \( C_i \) located at the site m. The cluster \( C_i \) has the schema \( S_i \).

\( TC_i^m[S_{ij}] \): denotes a temporary cluster \( ij \) located at site m and is the result of the join of \( i \)th and \( j \)th clusters.

\( TC_i^m[S_i] \): denotes a temporary cluster \( i \) located at site m and is result of the normal projection operation on the \( i \)th cluster.
Additionally, following notation is used in the formulation of the zero-one integer linear program for the query.

\( K_{\text{com}} \) represents the cost of transmitting one byte of data over the network.

\( K_{\text{io}} \) is the cost of fetching one block of data from the disk to the memory.

\( \text{BLK} \) is a size of disk block in bytes.

\[
\gamma_{i,j}^{m,n} = \begin{cases} 
1 & \text{if cluster } C^m_i \text{ and } C^n_j \text{ can be joined} \\
0 & \text{otherwise}
\end{cases}
\]

\[
\chi_{i,j}^{m,n} = \begin{cases} 
1 & \text{if cluster } C^m_i \text{ is sent to site } n \text{ and is joined with } C^n_j \\
0 & \text{otherwise}
\end{cases}
\]

The proposed query processing algorithm is heuristic based and its objective is to generate an efficient execution tree. The first step of the algorithm, commonly referred to as initial local processing, identifies all the local selections and projections. This step reduces the sizes of the base clusters. In the next step, the algorithm uses an iterative hill climbing procedure [7]. In each iteration of this procedure, the algorithm generates a list of all possible extended semi-joins and computes their profitability as defined in section 5.3.2. The most profitable semi-join is chosen as the next operation in the execution tree. Finally, the algorithm solves an integer linear program to generate the optimal sequence of the normal join operations. These steps are summarized in Figure 5-18 and explained in Section 5.3.1. The derived sequence of the operations and their respective processing sites are used to define the execution tree.
5.3.1 The Algorithm

Let Q be the query originating at site M and let \( A_u = \{ A_{u1}, A_{u2}, A_{u3}, \ldots, A_{uk} \} \) be the set of attributes requested by the user. The query, in general, has some restrictions (selections) placed on the clusters. Let \( A_s = \{ A_{s1}, A_{s2}, A_{s3}, \ldots, A_{sn} \} \) be the set of attributes involved in the selection operations. Let \( A_j = \{ A_{j1}, A_{j2}, A_{j3}, \ldots, A_{jm} \} \) be the set of attributes involved in the join operations. All the attributes are assumed to have unique names, in case of the ambiguity the names are qualified by the cluster names.

The set of the clusters referred to in the query Q is denoted by \( E_0 \) and defined as:

\[
E_0 = \{ C_i^n[ S_i ] \mid Attr( S_i ) \cap ( A_u \cup A_j \cup A_s ) \neq \emptyset \}
\]

The set \( E_0 \) contains all the clusters that have at least one attribute required for answering the query. The attributes required to answer the query are part of the sets \( A_u, A_s \) and \( A_j \).

Let \( E_0' \) be the set that contains the sizes of the corresponding clusters in set \( E_0 \).

\[
E_0' = \{ S \mid C_i^n[ S_i ] \mid C_i^n[ S_i ] \in E_0 \}
\]

**Initial Local Processing:**

The initial local processing specifies all the local selection operations expressed in the query. Next, it identifies all the attributes of each cluster that appear in the original query. Furthermore, the attributes that are needed to perform the required normal natural joins and do not explicitly appear in the query are also identified. The
Perform initial local processing, estimate sizes of new temporary clusters.
Repeat
{
  Compute cost, benefit and profit of all feasible extended semijoins.
  Select most profitable semijoin and append it to the execution tree.
  Modify the query graph to reflect the effect of the operation, estimate new sizes of temporary clusters.
}
Until no profitable semijoins are found.

Repeat
{
  Formulate integer linear program for the query graph.
  Solve the integer linear program, append the operations generated by the solution to the execution tree.
  Modify the query graph to reflect the effect of the operation, estimate new sizes of temporary clusters.
}
Until query graph has only one node.

Analyze the execution tree to detect and remove any redundant data movement.

Figure 5-18: The Proposed Query Processing Algorithm

Initial local processing subsequently generates the local projection operations for each cluster over the previously identified attributes in order to reduce the sizes of the base clusters. The steps to perform initial local processing phase of the algorithm are as follows:
1.1 Scan the query expression and generates all the local selection operations for each cluster.

1.2 Scan the query expression and generate all the local normal projection operations for each cluster in the query.

1.3 \( \forall C_i^m[S_i] \in E_0 \)

Compute \( A_i = \text{Attr}[S_i] \cap (A_u \cup A_j \cup A_s) \)

If \( A_i \neq \emptyset \) then, generate a normal projection operation \( \pi_{A_i}^n(C_i^m[S_i]) \), otherwise, drop cluster \( C_i^m \) from the set \( E_0 \) and from the corresponding size set \( E'_0 \).

1.4 Categorize the cluster is four subsets of \( E_{0a}, E_{0b}, E_{0c} \) and \( E_{0d} \) such that

\[
E_{0a} = \{ C_i^m[S_i] \mid \text{There are selection as well as projection operations on the cluster} \}
\]

\[
E_{0b} = \{ C_i^m[S_i] \mid \text{There are only projection operations on the cluster} \}
\]

\[
E_{0c} = \{ C_i^m[S_i] \mid \text{There are only selection operations on the cluster} \}
\]

\[
E_{0d} = E_0 - (E_{0a} \cup E_{0b} \cup E_{0c})
\]

1.5 Now process each set using the appropriate process described in the following sections:
Process $E_{0a}$: This procedure processes clusters in set $E_{0a}$. The process generates operations to perform simultaneous selection and projection operations on each of the cluster in the set. It adds these operations to the execution tree. The new set $E_{1a}$ built replacing $C_i^m[S_i] \in E_{0a}$ by an intermediate cluster $TC_i^m[S_j]$. Finally, it derives all the appropriate statistical profiles and compute the size of each cluster $TC_i^m[S_j]$.

$$E_{1a} = \{TC_i^m[S_j] | C_i^m[S_i] \in E_{0a}\}$$

$$E'_{1a} = \{SZ(TC_i^m[S_j]) | TC_i^m[S_j] \in E_{1a}\}$$

Process $E_{0b}$: $\forall C_i^m[S_i] \in E_{0b}$, the process adds all the projection operations to the execution tree. It also builds a new set $E_{1b}$ where each $C_i^m[S_i] \in E_{0b}$ is substituted by a temporary cluster $TC_i^m[S_j]$. Finally, the process derives the statistical information and sizes of the relevant clusters.

$$E_{1b} = \{TC_i^m[S_j] | C_i^m[S_i] \in E_{0b}\}$$

$$E'_{1b} = \{SZ(TC_i^m[S_j]) | TC_i^m[S_j] \in E_{1b}\}$$

Process $E_{0c}$: The process adds all the selection operations for the each of the cluster in set $E_{0c}$. It also estimates the stat-info and sizes
of the relevant intermediate clusters. Finally, it generates
new sets $E_{1c}$ where each $C_i^m[S_i] \in E_{oc}$ is substituted
by a temporary version of the cluster $TC_i^m[S_i]$.

$$E_{1c} = \{ C_{i(t1)}[S_j], C_i^m[S_i] \in E_{oc} \}$$

$$E'_{1c} = \{ Sz(C_{i(t1)}[S_j]), C_i^m[S_i] \in E_{1c} \}$$

**Process $E_{0d}$:** All the clusters in this set have no initial local processing
operations. This process copies the information in set $E_{0d}$
and $E_{0d}'$ to new sets $E_{1d}$ and $E_{1d}'$.

1.6. Generate set $E_1$ and $E_1'$ by computing the union of all the subsets.

$$E_1 = E_{1a} \cup E_{1b} \cup E_{1c} \cup E_{1d}$$

$$E'_1 = E'_{1a} \cup E'_{1b} \cup E'_{1c} \cup E'_{1d}$$

**Local Join Processing**

The normal join of two clusters residing at the same site can be performed
without incurring in any communications cost. Thus, in the next step the algorithm
performs all the local normal join operations. The steps of the local join processing
are as follows:

2.0 Generate the query graph [51] for the query.
2.1 Scan all the edges of the query graph. For each edge whose vertices (clusters) are on the same site. Generate an operation to perform local join of these two clusters. Insert the operation to the execution tree.

2.2 Derive relevant stat-info and compute the size of intermediate result cluster.

2.3 Modify the query graph to reflect the changes due to the join operations generated in the above steps.

2.4 Derive new set $E_2$ from the set $E_1$ by replacing the joined pair of clusters by their temporary result clusters. Also, generate corresponding size set $E_2'$ for the clusters in the set $E_2$.

2.5 If the query graph has no more edges left then the algorithm terminates. Otherwise, proceed to the extended semijoin processing step.

**Extended Semijoin Reduction Processing:**

The algorithm generates all possible extended semi-joins and computes their profitability. The most profitable extended semi-join is added to the execution tree. The sizes of the corresponding clusters are modified to reflect the estimated reduction using the estimation methodology presented in Section 5.2. The algorithm to generate the extended semijoin processing operations is described as follows:

3.0 Repeat 3.1 through 3.7 until there are no profitable semijoins.
3.1 Generate a set of all possible extended semijoins, \( S = \{ S_1, S_2, S_3, \ldots, S_n \} \), among the clusters in the query graph. A member of the set, \( S_i \), is an extended semijoin of any two permanent/temporary clusters, \( C_i^m/TC_i^m \) and \( C_j^n/TC_j^n \) provided the query graph has an edge between the vertices represented by these clusters.

3.2 Compute the cost, \( C_i \), incurred in performing the semijoin operation \( S_i \). The cost computation considers local processing costs as well. The dominating factor of the local processing cost is the disk access cost. i.e.,

\[
C_i = K_0 + K_{\text{com}} \cdot \text{sz}(\pi_a(C_j^n)) + K_{\text{io}} \cdot \left( \text{sz}(C_j^n)/BLK + \text{sz}(\pi_a(C_j^n))/BLK \right)
\]

3.3 The benefit of a semijoin is the reduction in the size of receiving cluster. The reduction, \( R_i \), in size of the receiving cluster and potential savings, \( B_i \), in future communication and disk access cost are computed as follows:

\[
R_i = \text{sz}(C_i^m) - \text{sz}(C_i^m \bowtie C_j^n)
\]

\[
B_i = K_{\text{com}} \cdot R_i + K_{\text{io}} \cdot R_i / BLK
\]

3.4 Compute the profit, \( P_i \), of the semijoin operation, \( S_i \).

\[
P_i = B_i - C_i
\]
3.5 Choose the semijoin \( S_m \) with maximum profit \( P_m \) and insert the semijoin operation into the execution tree.

3.6 Estimate the stat-info to reflect the impact of semijoin operation and compute the size of the new version of the cluster. Substitute the size of receiving cluster by the size of new version of the receiving cluster in the query graph.

**Join Optimization**

Typically, the semijoin based algorithms after the semijoin reduction phase collect all the reduced clusters at an assembly site. The final result is generated by locally joining all the temporary clusters at the assembly site. The cost of the assembly phase of a semijoin based algorithm is proportional to the sum of the sizes of the clusters transferred. The temporary clusters after the reduction phase may still have rows that will not be part of the result. The proposed algorithm generates a sequence of join operations after the semijoin reduction phase with the objective of reducing the total cost involved during the assembly phase of a semijoin based strategy. The proposed algorithm uses a zero-one integer linear programming formulation for generating the join strategy. The advantage of using an zero-one integer programming formulation is the availability of algorithms to solve the problem efficiently as long as the problem has a relatively small number of variables [54, 55, 57].

The formulation of the integer linear program for a query is given as follows:

The objective of the integer linear program is to determine the sequencing of the join operations in the distributed network so as to minimize the transmission and
the input-output processing costs. The objective function chooses those values of $X_{i,j}^{m,n}$ which minimize the objective function subject to the constraints specified here. The objective function consists of communication cost of sending a cluster $C_i^m$ over the network to the site at which it will be joined with cluster $C_n^l$, a future communication cost specified by the size of the result of the join operation between the cluster $C_i^m$ and cluster $C_n^l$ and the input-output processing cost of joining the two clusters.

Objective function:

$$
\min_{X_{i,j}^{m,n}} \left\{ \sum_{m,n=1}^{M} \sum_{i,j=1}^{N} K_{com} \cdot X_{i,j}^{m,n} (S_z(C_i^m) + S_z(C_n^l) N J N C_n^l) + \sum_{m,n=1}^{M} \sum_{i,j=1}^{N} K_{io} \cdot X_{i,j}^{m,n} (S_z(C_i^m) + S_z(C_n^l)) / BLK \right\}
$$

(67)

Subject to:

$$X_{i,j}^{m,n} + X_{j,i}^{n,m} = Y_{i,j}^{m,n}
$$

(68)

$$\sum_{m,n=1}^{M} \left( \sum_{i,j=1}^{N} X_{i,j}^{m,n} + \sum_{j=1}^{N} X_{j,i}^{n,m} \right) \leq 1 \quad i = 1, 2, \ldots, N
$$

(69)
\[
\sum_{m, n=1}^{M} \sum_{i, j=1}^{N} X_{i, j}^{m, n} \leq \xi
\]

where, \( \xi \) is the number of feasible joins at each iteration.

\[
X_{i, j}^{m, n} = (0, 1) \text{ for every } m, n, i, j
\]

The objective function chooses those values of \( X_{i, j}^{m, n} \) which minimize the sum of the processing, communication and the future communication cost of the results produced by the join operations. The first constraint (equation 68) on the model states that any two clusters can be joined only if there is a join condition involving these two clusters in the query. Also, they can be joined only once at a chosen site. The second constraint (equation 69) specifies that any cluster can be joined with only one other cluster in the network. This is due to the query materialization assumption. The third constraint (equation 70) specifies the maximum number of the normal join operations to be generated for each iteration of this model. If there are \( N \) clusters in the query graph then the \( \xi \) can have the value \( N/2 \) or \( (N + 1)/2 \) depending on whether \( N \) is an even or an odd number. The final constraint (equation 71) states that the variable \( X_{i, j}^{m, n} \) can have only zero or one as values. This constraint makes it a zero-one integer linear programming problem that can be solved using algorithms such as Balas zero-one additive algorithm [55]. The average case performance of this algorithm has been shown to be polynomial [55].
The solution to the above zero-one integer linear program generates a set of normal join operations which can be performed in parallel. These join operations are added to the execution tree.

The algorithm updates the query graph to reflect the impact of the normal join operations appended to the execution tree including the sizes of resulting temporary clusters. The zero-one integer linear program is reformulated every time the query graph is updated. This iterative process terminates when the final result is obtained. Finally, the complete execution tree is analyzed to detect and eliminate any redundant data movements.

**Global Semijoin Elimination**

At this stage, the algorithm examines all the extended semijoin operations in the execution tree. The semijoin reductions to the clusters that are not moved from the sites are eliminated.

**Loading Phase:**

Let $C_p$ be the final cluster generated at the end of the Join Optimization process. The cluster $C_p$ is the final temporary cluster at site p. The loading phase generates a projection operation to eliminate all the unwanted attributes from this cluster to produce the final result cluster $C_p$. Finally, it generates all the embed operations to structure the results in the schema desired by the user.
5.3.2 An Illustration

In this section, the execution of the query processing algorithm is illustrated using an example. The query represented by the query graph of Figure 5-13 is executed over the distributed heterogeneous database presented in the Figure 5-1.

The statistical profile (stat-info) of the base clusters is presumed to be stored along with the database. The information about the database and stat-info(s) is presented as follows.

`iueolog` is a relational cluster located at the IUE node of the network. The cluster is made up of a single table. The width of the table $\text{Sum}_1(iueolog) = 61$ bytes and total number of rows in table $\text{tuple}_1(iueolog) = 25$. The size of this cluster $\text{Sz}(iueolog) = 25 \times 61 = 1525$ bytes. The stat-info for the objid and the img attributes along with their association matrix is presented as follows:

Stat-info for the attribute objid is a list of following $<\text{attribute value}, c_{\text{row}}>$ pairs.

$<\text{or_nebul},5> <\text{jupiter},4> <\text{m42area},4> <\text{nova_vul},4> <\text{moon},4> <\text{crabneb},4>$

Stat-info for the key attribute img is a vector of 32 bits.

$[11111111111111110111100000101]$  

The association information between the objid and img attributes is stored as pair of objid values and corresponding bit vectors.
Sao is a network cluster located at the CFA (Center for Astrophysics) node of the network. Sao cluster consists of three tables field (1), sources (2), instrument (3).

\[
\begin{align*}
\text{sum}_1(\text{Sao}) &= 26 & \text{tuple}_1(\text{Sao}) &= 3 \\
\text{sum}_2(\text{Sao}) &= 26 & \text{tuple}_2(\text{Sao}) &= 6 \\
\text{sum}_3(\text{Sao}) &= 5 & \text{tuple}_3(\text{Sao}) &= 9
\end{align*}
\]
\[
\text{Sz}(\text{sao}) = 26*3 + 26*6 + 5*9 = 279 \text{ bytes}
\]

The stat-info for objid attribute is a 5-tuple \(<\text{value},\text{c}_\text{row},\text{trow1},\text{trow2},\text{trow3}>\) and is given as follows:

\[
\begin{align*}
<\text{or}_\text{nebul},4,4,1,2> & \quad <\text{m42area},4,4,1,2> & \quad <\text{cluster},2,2,1,2> \\
<\text{moon},2,2,1,1> & \quad <\text{jupiter},2,2,1,1> & \quad <\text{hd2151},2,2,1,1>
\end{align*}
\]

Irascce is a hierarchical cluster located at the IPAC (infrared processing and analysis center) node of the network. PSC cluster has two tables Psc(1) and observation(2).
\[ \text{sum}_1(\text{psc}) = 34 \quad \text{tuple}_1(\text{psc}) = 7 \]
\[ \text{sum}_2(\text{observation}) = 20 \quad \text{tuple}_2(\text{observation}) = 17 \]
\[ \text{sum}_1(\text{Sao}) = 5 \quad \text{tuple}_1(\text{Sao}) = 9 \]
\[ \text{Sz}(\text{Iraspsc}) = 34 \times 7 + 20 \times 17 = 578 \text{ bytes} \]

The stat-info for flux is 2-tuple \(<\text{value},\text{crow}>\) and is given as follows:

\[ <18,4> \ <23,5> \ <28,2> \ <42,4> \ <50,2> \]

The stat-info for wave attribute is 2-tuple \(<\text{value},\text{crow}>\) and is provided below.

\[ <10,5> \ <15,5> \ <20,3> \ <25,4> \]

The stat-info for objid attribute is 4-tuple \(<\text{value},\text{crow},\text{trow1},\text{trow2}>\) and is provided below.

\[ <\text{or_nebul},3,3,1> \ <\text{jupiter},6,6,1> \ <\text{m42area},1,1,1> \]
\[ <\text{moon},4,4,1> \ <\text{nova_vul},2,2,1> \ <\text{md11a},1,1,1> \]

The association information between the wav and the flux attributes is stored as 3-tuples \(<\text{value(wav)},\text{value(flux)},\text{crow}>\). For the Iraspsc cluster information is given as follows:

\[ <10,18,2> \ <10,23,1> \ <10,28,2> \ <15,18,1> \ <15,23,3> \ <15,50,1> \]
\[ <20,23,2> \ <20,42,1> \ <25,42,3> \ <25,50,1> \]

The association information between the wav and the objid attributes is stored as 3-tuples \(<\text{value(wav)},\text{value(objid)},\text{crow}>\). For the Iraspsc cluster information is provided as follows:
<10, jupiter, 2> <10, moon, 1> <10, nova_vul, 1> <10, md11a, 1>
<15, or_nebul, 1> <15, jupiter, 1> <15, m42area, 1> <15, moon, 1>
<15, nova_vul, 1> <20, or_nebul, 1> <20, jupiter, 1> <20, moon, 1>
<25, or_nebul, 1> <25, jupiter, 2> <25, moon, 1>

For the sample query Q described by the query graph of Figure 5-13 the attribute sets are as follows:

\[ A_u = \{ \text{Iuelog.ra, Iuelog.dec, Iuelog.objid, img, image\_name} \} \]
\[ A_s = \{ \text{wav, flux} \} \]
\[ A_j = \{ \text{Iuelog.objid, Iuelog.img, Iraspse.objid, Sao.objid, img} \} \]

Also, for ease of presentation the sites IPAC, CFA, IUE, IUEIM are denoted by 1, 2, 3, 4, respectively and clusters Iraspse, Sao, Iuelog and Pilog are denoted by the \( C_1, C_2, C_3 \) and \( C_4 \) respectively. Hence the cluster Iraspse located at IPAC site is \( C_1^1 \). The initial sets for the query are given as follows:

\[ E_0 = \{ C_1^1, C_2^2, C_3^3, C_4^4 \} \]
\[ E_0' = \{ 578, 261, 1525, 1280 \} \]

**Initial Local Processing**

\( C_1^1 \) is the only cluster for which simultaneous selection projection operation is feasible and necessary.

\[ E_{o\alpha} = \{ C_1^1 \} \]
\[ E_{o\alpha}' = \{ 578 \} \]
For $C^2_2$, $C^3_3$, $C^4_4$ clusters, normal projection operations are necessary and feasible.

\[
E_{ob} = \{ C^2_2, C^3_3, C^4_4 \} \\
E'_{ob} = \{ 261, 1525, 1280 \} \\
E_{oc} = \text{null} \\
E_{oa} = \text{null}
\]

**Process $E_{0a}$:**

Add following selection projection operation in the execution plan:

\[
\pi_{objid} (\sigma_{wav<10} (\sigma_{flux>10} (C^1_1)))
\]

The size of result cluster $TC^1_1$ is estimated by using the stat-info technique. The derived stat-info of 'objid' attribute in $TC^1_1$ is:

\[
<jupiter,2> <moon,1> <nova_vul,1> <md11a,1>
\]

Thus,

\[
E_{1a} = \{ TC^1_1 \} \\
E'_{1a} = \{ 50 \}
\]

**Process $E_{0b}$:**

Add following normal projection operations in the execution plan:
\[ \pi_{\text{objid}}(C_2), \quad \pi_{\text{ra,dec,imag.objid}}(C_3), \quad \pi_{\text{image_num,image_ptr}}(C_4) \]

The sizes of the result clusters of these operations can be estimated by manipulating the stat-info.

The stat-info of objid in \( TC_2^2 \) cluster is:

\(<\text{or\_nebul},1> <\text{m42area},1> <\text{cluster},1>\)
\(<\text{moon},1> <\text{jupiter},1> <\text{hd2151},1>\)

and width of a row is 10 bytes. Thus \( \text{sz}(TC_2^2) = 10 \times 6 = 60 \) bytes

The stat-info of attributes in clusters \( TC_3^3 \), \( TC_4^4 \) remain unchanged. But the width of a row for the clusters \( TC_3^3 \) and \( TC_4^4 \) is modified to 22 and 28 respectively. Hence the sizes of clusters are as follows:

\[ \text{Sz}(TC_3^3) = 22 \times 25 = 550 \]
\[ \text{Sz}(TC_4^4) = 28 \times 40 = 1120 \]

\[ E_{18} = \{ TC_2^2, \; TC_3^3, \; TC_4^4 \} \]
\[ E'_{18} = \{ 60, \; 550, \; 1120 \} \]

The final result of performing the initial local processing of step 1 is given as follows:

\[ E_1 = \{ TC_1^1, \; TC_2^2, \; TC_3^3, \; TC_4^4 \} \]
\[ E'_1 = \{ 50, \; 60, \; 550, \; 1120 \} \]
**Local Join Processing**

The query does not have any feasible local joins. Hence

\[ E_2 = E_1 \quad \text{and} \quad E_2' = E_1' \]

**Extended Semijoin Reduction Phase**

The estimated cost and benefits of all possible semijoins are summarized as follows.

<table>
<thead>
<tr>
<th>No</th>
<th>Semijoin</th>
<th>Cost</th>
<th>Benefit</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$TC_1^1 \times \text{objid}</td>
<td>TC_2^2$</td>
<td>65</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>$TC_2^2 \times \text{objid}</td>
<td>TC_1^1$</td>
<td>45</td>
<td>40</td>
</tr>
<tr>
<td>3</td>
<td>$TC_2^2 \times \text{objid}</td>
<td>TC_3^3$</td>
<td>65</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>$TC_3^3 \times \text{objid}</td>
<td>TC_2^4$</td>
<td>65</td>
<td>176</td>
</tr>
<tr>
<td>5</td>
<td>$TC_3^3 \times \text{img}</td>
<td>TC_4^4$</td>
<td>165</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>$TC_4^4 \times \text{img}</td>
<td>TC_3^3$</td>
<td>105</td>
<td>420</td>
</tr>
</tbody>
</table>

The semijoin (6) is the most profitable semijoin. Add this operation to the execution plan. The relevant stat-info for $TC_4^4$ is computed and the estimated size of temporary cluster is 700.
\[ E_3 = \{ TC_1^1, TC_2^2, TC_3^3, TC_4^4 \} \]
\[ E'_3 = \{ 50, 60, 550, 700 \} \]

The estimated cost/benefits for the next iteration of the step are summarized as follows:

<table>
<thead>
<tr>
<th>No</th>
<th>Semijoin</th>
<th>Cost</th>
<th>Benefit</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TC_1^1 &lt; objid]TC_2^2</td>
<td>65</td>
<td>10</td>
<td>-55</td>
</tr>
<tr>
<td>2</td>
<td>TC_2^2 &lt; objid]TC_1^1</td>
<td>45</td>
<td>40</td>
<td>-5</td>
</tr>
<tr>
<td>3</td>
<td>TC_2^2 &lt; objid]TC_3^3</td>
<td>65</td>
<td>20</td>
<td>-45</td>
</tr>
<tr>
<td>4</td>
<td>TC_3^3 &lt; objid]TC_2^2</td>
<td>65</td>
<td>176</td>
<td>111</td>
</tr>
<tr>
<td>5</td>
<td>TC_3^3 &lt; img]TC_4^4</td>
<td>105</td>
<td>0</td>
<td>-105</td>
</tr>
<tr>
<td>6</td>
<td>TC_4^4 &lt; img]TC_3^3</td>
<td>105</td>
<td>0</td>
<td>-105</td>
</tr>
</tbody>
</table>

The semijoin (4) is the most profitable semijoin. Add this operation to the execution plan. The relevant stat-info for TC_4^4 is computed and its estimated size is 374.

\[ E_3 = \{ TC_1^1, TC_2^2, TC_3^3, TC_4^4 \} \]
\[ E'_3 = \{ 50, 60, 374, 700 \} \]

Join Optimization Phase
Following is the integer linear programming formulation for the reduced query.

\[ 50X_{1,2}^{1,2} + 20X_{1,2}^{1,2} + 60X_{2,1}^{2,1} + 20X_{2,1}^{2,1} \]
\[ + 60X_{2,3}^{2,3} + 374X_{2,3}^{2,3} + 374X_{3,2}^{3,2} + 374X_{3,2}^{3,2} \]
\[ + 374X_{3,4}^{3,4} + 782X_{3,4}^{3,4} + 700X_{4,3}^{4,3} + 782X_{4,3}^{4,3} \]

Subject to

\[ X_{1,2}^{1,2} + X_{2,1}^{2,1} \leq 1 \]
\[ X_{2,3}^{2,3} + X_{3,2}^{3,2} \leq 1 \]
\[ X_{3,4}^{3,4} + X_{4,3}^{4,3} \leq 1 \]
\[ X_{1,2}^{1,2} + X_{2,3}^{2,3} + X_{2,1}^{2,1} + X_{3,2}^{3,2} \leq 1 \]
\[ X_{2,3}^{2,3} + X_{3,4}^{3,4} + X_{3,2}^{3,2} + X_{4,3}^{4,3} \leq 1 \]
\[ X_{3,4}^{3,4} + X_{4,3}^{4,3} \leq 1 \]
\[ X_{1,2}^{1,2} + X_{2,1}^{2,1} + X_{2,3}^{2,3} + X_{3,2}^{3,2} + X_{3,4}^{3,4} + X_{4,3}^{4,3} = 1 \]

Using Lindo [54] the following solution was obtained for the formulation.

\[ X_{1,2}^{1,2} = 1 \quad X_{2,1}^{2,1} = 0 \]
\[ X_{2,3}^{2,3} = 0 \quad X_{3,2}^{3,2} = 0 \]
\[ X_{3,4}^{3,4} = 0 \quad X_{4,3}^{4,3} = 0 \]
The normal join operation corresponding to the variable $X^{1,2}_{1,2}$ is added to the execution plan. The estimated size of the result cluster of this operation is 30 and the query sets reduce to:

$$E_4 = \{ TC_{12}^2, TC_{3}^3, TC_4^4 \}$$
$$E_4' = \{ 20, 374, 700 \}$$

Following is the integer linear program for the reduced problem.

$$20X_{12,3}^{2,3} + 176X_{12,3}^{2,3} + 374X_{3,12}^{3,2} + 176X_{3,12}^{3,2}$$
$$+ 374X_{3,4}^{3,4} + 782X_{3,4}^{3,4} + 700X_{4,3}^{4,3} + 782X_{4,3}^{4,3}$$

Subject to

$$X_{12,3}^{2,3} + X_{3,12}^{3,2} \leq 1$$
$$X_{3,4}^{3,4} + X_{4,3}^{4,3} \leq 1$$

$$X_{12,3}^{2,3} + X_{3,12}^{3,2} + X_{3,4}^{3,4} + X_{4,3}^{4,3} \leq 1$$

$$X_{12,3}^{2,3} + X_{3,12}^{3,2} + X_{3,4}^{3,4} + X_{4,3}^{4,3} = 1$$

Using Lindo [54] the following solution was obtained for the formulation.

$$X_{12,3}^{2,3} = 1 \quad X_{3,12}^{3,2} = 0$$
$$X_{3,4}^{3,4} = 0 \quad X_{4,3}^{4,3} = 0$$
Add normal join operation corresponding to variables which have value of 1. The estimated size of the result cluster of the join operation is 176. The query set reduces to:

\[
E_4 = \{ \, TC_{123}^3, \, TC_4^4 \, \}
\]
\[
E'_4 = \{ \, 176, \, 700 \, \}
\]

The next step of algorithm adds a normal join operation of the remaining two clusters at site 4. The cost incurred at this step is 176.

\[
E_4 = \{ \, TC_{1234}^4 \, \}
\]
\[
E'_4 = \{ \, 368 \, \}
\]

**Global Semijoin Elimination Phase**

The cluster \( C_3 \) and \( C_4 \) do not move from their sites before being joined with the other clusters. Thus, semijoin reduction operations corresponding to these clusters are eliminated from the query plan.

**Loading Phase**

The final cluster resulting from the join optimization phase has all the attributes in user desired schema. Hence, no operations are added to the execution tree.
5.3.3 Conclusion

The algorithms described for the statistical estimation and the query processing for a heterogeneous distributed databases were implemented in the "C" language on VAX/VMS and Sun/Unix environments. The implementation details and test results are described in the following chapter. The complete code can be found in [68].
6 Implementation and Experience

The query processing and related algorithms proposed in the preceding chapters were implemented as a part of the query processing module of the DAVID heterogeneous distributed database management system. The sub-modules of the query processing system of the DAVID system are shown in Figure 6-1.

The GSQL queries received by the DAVID system go through a sequence of processing steps including the syntax checking, the semantics checking, the privilege checking, the complex to basic translation, the virtual to conceptual query translations and the query decomposition. The semantic checking module retrieves various cluster definitions from the dictionary and the estimation module retrieves stat-info from the StatsDB. The final product of the query processing module is a query evaluation plan. The query evaluation plan is also referred to as the execution tree. Each node of the execution tree is an operation of the primitive type. The queries received and processed by the system fall into three categories.

(1) **Primitive Type:** These queries involve only one cluster resident in a DBMS, also the result of the query is directed to the same site where the source cluster resides.

(2) **Basic Type:** These queries may involve more than one cluster, one site or the virtual clusters. Thus, these queries require query translation and decomposition to generate the execution tree.
Figure 6-1: Query Processing Sub-modules
(3) Complex Type: These queries require more than one pass to evaluate. The example of these queries include the nested queries, the union queries and the aggregate queries. A complex query requires additional translation step to generate a tree of basic queries.

The complex_to_basic module translates a complex query to a tree of basic queries. The module analyzes the input query for the type of complexity. The complexity could be of the nested, union and aggregate type. Depending upon the type nested_to_basic, union_to_basic or function_to_basic sub-modules are invoked. The final result of the module is a tree of basic queries.

The syntax checking module checks for the correctness of the query syntax. The semantic and privilege checking sub module ensure that the user has authority to access the data and also, the data objects referred to in the query are defined and accessible to the system. Also, this module generates the schema definition of the result cluster.

The virtual_to_conceptual translation module implements the algorithm proposed in chapter 4 of this work. The algorithm processes a tree of basic queries. Each basic query is scanned for the virtual cluster references. Each virtual cluster referred in the cluster algebra query is replaced by its "mapping" cluster algebra expression. The algorithm translates a query with the virtual cluster references to an equivalent query that is expressed over the conceptual clusters only.

The estimation module is made up of several procedures. Each procedure corresponds to a cluster algebra operation and estimates the impact of actually
executing the operation. These procedures estimate the stat-info of a specified attribute of the result cluster without actually executing the operation. The estimation procedures are described as follows:

The \texttt{estimate\_select} procedure accepts the source cluster, the condition and the stat-info of the attribute in the condition. The output of the function is the stat-info of the condition attribute for the result cluster. It uses the algorithm described in Section 5.2.1.3.1 for deriving the output stat-info.

The \texttt{estimate\_project} procedure derives stat-info of the specified attributes of the output cluster using their stat-info for the input cluster. The stat-info is derived using the algorithm presented in Section 5.2.1.3.2.

The \texttt{estimate\_join} procedure estimates the stat-info of an attribute that is part of the join condition. It uses the stat-info of the attribute of clusters being joined and derives the resulting stat-info. The procedure implements the algorithm described in Section 5.2.1.3.3.

The \texttt{estimate\_semijoin} procedure uses stat-infos of the condition attribute in the receiving and sending clusters. The stat-info of the attribute of the receiving cluster is derived using the algorithm described in Section 5.2.1.3.4. The procedure estimates the stat-info of the condition attribute in the reduced receiving cluster.

The stat-info derivation procedures described above estimate the stat-info of the condition attributes only. These attributes are also referred to as explicit attributes. The stat-info of the other attributes that are not part of the conditions is estimated by the \texttt{estimate\_implicit\_stat-info} procedure. The procedure assumes that
the stat-info of a condition attribute has been derived using one of the above four procedures. The procedure uses the association matrix information between the condition attribute and the implicit attribute. The procedure implements the algorithm described in Section 5.2.1.3.5. The output of the procedure are the estimated stat-info of the implicit attribute in the result cluster and also the association matrix between these two attributes of the result cluster.

The algorithm described in Section 5.3.1 is implemented for the decomposition of the basic queries. The implementation of the algorithm consists of the query graph generator (QG_Gen), the initial local processor (ILP), the local join generator (LJN_Gen), the semijoin generator (SJN_Gen), the integer linear program formatter (IP_Gen) and the post analysis processor (Post_analyzer). The data flow diagram of the implemented algorithm is depicted in Figure 6-2.

The query graph generator (QG_Gen) procedure parses a GSQL query and generates a query graph data structure [51] for the query. Vertices of the query graph represent the source clusters of the query. An edge between two vertices of the query graph denotes a join operation. The label of an edge is the join operation itself. All the selection and projection operations on a clusters are attached to the vertex corresponding to the cluster.

The initial local processor (ILP) procedure scans all the vertices of the query graph. For each vertex it generates all possible local selection and projection operations and adds them to the execution tree. The procedure implements the initial local processing step of the algorithm described in Section 5.3.1. It modifies the query
Figure 6-2: Components of Query Decomposition Algorithm

graph to account for the impact of operations added to the execution tree. The algorithm calls appropriate estimation procedures to derive the statistical information of the required attributes.
The *local join generator* (*LJN_Gen*) procedure implements the local join processing step of the algorithm. It generates all possible normal join operations that can be performed without any data transfer. The procedure examines all the edges of the query graph. For an edge whose both vertex clusters reside at the same node a corresponding normal join operation is added to the execution tree. The two vertices are merged into one in the query graph. The sizes and statistical information of the appropriate clusters is estimated by calling the *estimate_join* procedure.

The *semijoin generation* (*SIN_Gen*) procedure iteratively generates all possible profitable semijoin operations for a query. The procedure generates a list of all possible semijoin operations among the vertices of the query graph. It then estimates the cost and benefit for each semijoin operation. The most profitable operation from the list is chosen and added to the execution tree. The algorithm then modifies the query graph and estimates the sizes and statistical information of the relevant clusters, by calling the *estimate_semi* procedure, to account for the operation added into the execution tree. The procedure iterates until no profitable semijoins could be found. The procedure implements the extended semijoin processing step of the algorithm described in Section 5.3.1.

The *integer linear program formulator* (*IP_Gen*) procedure formulates a zero-one integer linear program for the query graph. The process of formulation is described in the join optimization step of the algorithm defined in Section 5.3.1. The zero-one integer linear program is then submitted to a linear program solver such as LINDO [56]. The operations corresponding to the solution are added to the execution tree. The query graph is suitably modified to reflect the impact of operations added to the
execution tree. The sizes and statistical information of the relevant clusters is estimated by calling the estimate_join procedure. If the modified query graph has more than one vertex then the IP_Gen is called iteratively. Otherwise, the procedure terminates.

The post analysis (Post_analyzer) procedure analyzes the execution tree and eliminates any redundant operations from the execution tree. The semijoin reductions of the clusters that are not moved from their sites are eliminated from the execution tree. This procedure implements the global semijoin elimination step of the algorithm described in Section 5.3.1.

The final execution tree resulting from the query decomposition algorithm is then submitted to the DB System server (Figure 2-3) of the DAVID system for evaluation purposes.

6.1 Comparison and Analysis of Results

The heterogeneous distributed database management prototypes that perform global query processing include the Multibase, the PRECI, the Mermaid, the IMDAS and the DAVID. The implemented versions of the PRECI, Mermaid and the IMDAS deal with the heterogeneity of the relational databases only.

The query processing of the Multibase support access to the relational, hierarchical and network data models. The query processor of the Multibase decomposes a query into a query processing strategy. The query processing strategy
consists of a set of sub-queries, each of the sub-query references exactly one local schema; a set of transfer operations, to move the results of these sub-queries to the site of the query processor; and a set of operations to assemble the final result.

The query processing algorithm implemented for the DAVID translates the query into an internal representation called query graph. The algorithm generates initial local processing operations to reduce the data movement for the query. In the initial local processing step, it generates the local selection and projection operations on databases stored at each individual site. In the next step, it generates operations to move all the data to the user query site. In the final step, it assembles the answer from the transferred data.

The global data integration and uniform access to the data stored in relational, hierarchical and network data models were the major objectives of these systems. The idea was to implement a system where a query, that accesses data stored under different data models, can be expressed in a common language and answered. The issue of efficient query processing for the heterogeneous databases to answer a query became important only after the above mentioned objectives of these systems were met. Since the implemented query processing algorithms of these systems are very primitive, any algorithm that performs semijoin reductions or join sequencing will be an improvement over the existing algorithms. A query decomposition algorithm based on the semijoin reductions was proposed for the DAVID system [47]. This algorithm used selectivity approach based on uniform distribution for the purposes of estimating the temporary result sizes. The query decomposition algorithm developed and described in this work, herein called QD_New, used the initial local
processing, the semijoin reductions followed by the zero-one integer linear programming based join sequencing. The QD_New algorithm will predictably produce execution trees with lesser or equal cost of evaluation for a query. This is due to the fact that it uses a provably accurate methodology (refer to propositions 4.1, 4.2 and 4.3) for estimating the cost of evaluation for the operations in the execution tree. The semijoin reduction strategy is embedded in the QD_New algorithm. In addition, the QD_New finds an optimal sequence of the join operations to assemble and load the result of a query compared to the fixed strategy of moving all the data to the site containing the maximum data used by the semijoin based algorithms. In order to confirm the benefit provided by the join sequencing step of the QD_New, the semijoin based algorithm was modified and implemented to use the estimation methodology developed as a part of this work and used by the QD_New algorithm. The semijoin based algorithm with the non-parametric estimation methodology to estimate the sizes of intermediate results is referred to as QD_SJN.

The query decomposition algorithm (QD_New) presented above was compared to a semijoin based query decomposition algorithm (QD_SJN) also implemented for the DAVID system. The QD_SJN algorithm decomposes the user query and generates an execution tree. The decomposition steps of the QD_SJN consist of initial local processing, local join generation, semijoin reductions followed by assembly and loading steps. In the assembly step, all the reduced and required clusters after the semijoin phase are transferred to the site containing the maximum data. In the loading step, all the transferred clusters are joined to produce the final result. The QD_New differs from the QD_SJN in the assembly and the loading
phase. The QD_new algorithm formulates a zero-one integer linear program to
determine the best sequence of transfer and join operations instead of the assembly
and loading phase. In the query processing over the geographically distributed
networks the data transmission cost is the dominating factor [47]. Thus, the number
of bytes transmitted over the network to answer a query was used as a measure of
the cost of processing the query. This means that the constants $K_{\alpha}, K_{\lambda_0}$ were set to
zero for the purpose of computing the cost, benefit and objective functions. The
constant $K_{com}$ was set to one. The cost, benefit and objective functions with these
constant are as follows:

Cost function

$$C_i = \sum_{j} S_z(\pi_{\alpha}(C_{j}^n))$$

Benefit function

$$B_i = \sum_{i} S_z(C_{i}^n) - \sum_{i} S_z(C_{i}^m) - \sum_{j} S_z(C_{j}^n)$$

Objective function

$$\min_{X_{i,m,j}^n} \left\{ \sum_{m, n=1}^{M} \sum_{i,j=1}^{N} X_{i,m,j}^n(S_z(C_{i}^m) + S_z(C_{i}^n) \cdot N J N \cdot C_{j}^n) \right\}$$

Both of these algorithm were used for generating the execution trees for queries
expressed over the sample astrophysics databases described in Figure 5-1. Ten of
the queries used for evaluation are expressed as follows:
Q1: create actual cluster .temp.xx(1)
    select t.ra, t.dec, t.objid, u.imagename
    from iuesn1.david.iue.iuelog(1) t, nssdc.david.iue.pilog(1) u
    where t.img = u.img;

Q2: create actual cluster .temp.xx(1)
    select r.ra, r.dec, r.objid, s.cl
    from ipac.david.iras.iraspse(1) r, cfa.david.sao.sao(1) s
    where r.objid = s.objid;

Q3: create actual cluster .temp.xx(1)
    select r.ra, r.dec, r.objid, s.cl
    from ipac.david.iras.iraspse(1) r, cfa.david.sao.sao(1) s
    where r.objid = s.objid and s.cl = 72;

Q4: create actual cluster .temp.xx(1)
    select t.ra, t.dec, t.objid, u.imagename
    from cfa.david.sao.sao(1) s,
        iuesn1.david.iue.iuelog(1) t, nssdc.david.iue.pilog(1) u
    where s.objid = t.objid and t.img = u.img;

Q5: create actual cluster .temp.xx(1)
    select t.ra, t.dec, t.objid, u.imagename
    from ipac.david.iras.iraspse(1) r,
        iuesn1.david.iue.iuelog(1) t, nssdc.david.iue.pilog(1) u
    where r.wave = 15
and r.objid = t.objid and t.img = u.img;

Q6: create actual cluster .temp.xx(1)
select t.ra, t.dec, t.objid, u.imagename
from ipac.david.iras.iraspsc(1) r,
 iuesn1.david.iue.iuelog(1) t, nssdc.david.iue.pilog(1) u
where r.objid = t.objid and t.img = u.img;

Q7: create actual cluster .temp.xx(1)
select t.ra, t.dec, t.objid, u.imagename
from ipac.david.iras.iraspsc(1) r, cfa.david.sao.sao(1) s,
 iuesn1.david.iue.iuelog(1) t, nssdc.david.iue.pilog(1) u
where r.objid = s.objid
    and s.objid = t.objid and t.img = u.img;

Q8: create actual cluster .temp.xx(1)
select t.ra, t.dec, t.objid, u.imagename
from ipac.david.iras.iraspsc(1) r, cfa.david.sao.sao(1) s,
 iuesn1.david.iue.iuelog(1) t, nssdc.david.iue.pilog(1) u
where r.wave=10 and r.objid = s.objid
    and s.objid = t.objid and t.img = u.img;

Q9: create actual cluster .temp.xx(1)
select t.ra, t.dec, t.objid, u.imagename
from ipac.david.iras.iraspsc(1) r, cfa.david.sao.sao(1) s,
iatesn1.david.iue.iuelog(1) u
where r.flux > 10 and r.wave = 10 and r.objid = s.objid
    and s.objid = t.objid and t.img = u.img;

Q10: create actual cluster .temp.xx(1)
    select t.ra, t.dec, t.objid, t.class
    from ipac.david.iras.iraspsc(1) r, cfa.david.sao.sao(1) s,
    iuesn1.david.iue.iuelog(1) t
    where r.wave = 10 and r.objid = s.objid
    and s.objid = t.objid;

The execution trees generated by the algorithms were executed to find the actual cost of answering a query. Since, the communication cost dominates the distributed processing, the number of bytes of data transferred over the network was used as the cost of answering a query. The following table summarizes the results obtained by both algorithms:
<table>
<thead>
<tr>
<th>Query</th>
<th>No. Of Join operations</th>
<th>No. Of bytes transmitted for QD_SJN</th>
<th>No. Of bytes transmitted for QD_New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>1</td>
<td>550</td>
<td>550</td>
</tr>
<tr>
<td>Q2</td>
<td>1</td>
<td>122</td>
<td>122</td>
</tr>
<tr>
<td>Q3</td>
<td>1</td>
<td>28</td>
<td>28</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>424</td>
<td>424</td>
</tr>
<tr>
<td>Q5</td>
<td>2</td>
<td>620</td>
<td>532</td>
</tr>
<tr>
<td>Q6</td>
<td>2</td>
<td>610</td>
<td>522</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>604</td>
<td>534</td>
</tr>
<tr>
<td>Q8</td>
<td>3</td>
<td>564</td>
<td>284</td>
</tr>
<tr>
<td>Q9</td>
<td>3</td>
<td>564</td>
<td>284</td>
</tr>
<tr>
<td>Q10</td>
<td>2</td>
<td>130</td>
<td>88</td>
</tr>
</tbody>
</table>

The queries used for comparison purposes consisted of varying number of source clusters and the join operations. The test set consisted of the queries involving the most frequently used attributes by scientists on the ADS [58]. The proposed algorithm and the semijoin based algorithm produced identical results for the queries involving two sources clusters and one join operation. This can be explained from the fact that the proposed algorithm also uses semijoin reduction as one of the step in the processing. In case of two clusters, there is only one joining strategy i.e. moving
the cluster with smaller size to the site of the cluster with larger size. The semijoin based algorithm does exactly the same thing during the assembly phase. It moves all the reduced clusters to the site with maximum data. In case of queries involving more than two source clusters and one join operation, the proposed algorithm produced a strategy either with lower or equal cost. In our test set of queries the proposed algorithm produced a better strategy in 85% of the cases. The semijoin based algorithm used by DAVID does not reduce a cluster if the cost of performing reduction outweighs the benefit. Also, the semijoin operation does not eliminate all the unwanted rows from both sending and receiving clusters. In all such cases, the proposed algorithm generates a join strategy whose total cost is lower than the assembly phase of the semijoin based algorithm. In cases where the semijoin based algorithm fully reduces all the clusters in a query, both algorithms generate strategies with the same cost of evaluation.
7 Conclusion and Future Directions

The goal of this research was to provide a uniform query processing capability in a system of heterogeneous databases. The objective required selection of a general data model capable of integrating the heterogeneous databases, an uniform query language for expressing ad-hoc queries on the heterogeneous databases, a low level query language for expressing the primitive operations on heterogeneous databases, an statistical estimation methodology for the heterogeneous databases and finally, a query decomposition algorithm capable of generating good execution trees for the queries expressed over heterogeneous databases.

The cluster data model based on database logic was selected and utilized for representing and integrating heterogeneous databases. The integration of the heterogeneous databases was done by defining the virtual clusters. The Generalized Structured Query Language (GSQL) was selected for expressing the ad-hoc queries.

As a part of this work, a low level query language called cluster algebra was developed to express the primitive operations on heterogeneous databases. The cluster algebra is a generalization of the relational algebra. The cluster algebra operators are defined for the three common data models, i.e., relational, hierarchical and network models. The cluster algebra consists of the selection, normal projection, normal join, extended semijoin, union, difference, intersection, embed and unembed operations. The selection, union, difference and intersection operations were extended in their scope so that they can be applied to the data stored in any of the
three common data models. The generalization of the scope of the projection and join operation to the heterogeneous databases leads to the ambiguous results [4,42]. The "normal projection" and "normal join" operations, to perform the same functions with unambiguous results for the three common data model, were formulated in this work. The embed and unembed operations, to transform the databases from the one to the another of the three common data model, were introduced in this work. A grammar to formulate the cluster algebra queries was also developed in this work.

In this work, an algorithm to translate the cluster algebra queries expressed over the virtual views to the equivalent cluster algebra queries that are expressed over the conceptual databases was also developed and implemented.

This work also introduced a non-parametric estimation methodology for the heterogeneous databases. The methodology developed here estimates the result sizes of the cluster algebra operations expressed over the base as well as the temporary databases generated as a part of the query decomposition process. The implemented estimation methods were used by both the QD_New and QD_SJN decomposition algorithms. The algorithms were also analyzed for the accuracy. Propositions 4.1, 4.2 and 4.3 showed that the methodology produces accurate estimates for the selection, normal join and semijoin operations.

Finally, the research proposed, developed and implemented a query decomposition algorithm for the heterogeneous distributed database management systems. The objective of the query decomposition algorithm was to minimize the processing cost. The processing cost is made up of both the data transmission and local processing cost. In heterogeneous distributed database management systems
running over wide area networks the data transmission dominates the overall cost [7,17,24,51,56]. The transmission cost can be minimized by reducing the amount of data moved among the sites [7,9,13,17,24,36,51,56]. In the first step, the proposed algorithm generates operations to perform initial local processing and local join operations as these steps involve no data transfer. In the next step, the algorithm generates operations to perform profitable semijoin reductions. In the final step, the algorithm formulates a zero-one integer linear program to find an optimal sequence of join operations to assemble and load the result. Multibase and DAVID are the only two systems capable of processing a query expressed over the data stored in the three common data models. The query decomposition algorithms for these systems perform initial local processing, possible semijoin reductions, followed by assembly and loading of the result. The QDNuevo algorithm was implemented in the context of the DAVID system. The QDNuevo algorithm performs initial local processing, possible semijoin reductions, followed by the integer linear program formulation to find the optimal sequence of the join operation to load the result. The algorithm that uses only semijoin reductions followed by the assembly and loading of the result, with the same estimation methodology, called QD_SJN was also implemented in the context of the DAVID system. The preliminary comparison of these two algorithms showed that the QDNuevo algorithm reduced the data transmission in most of the queries involving three or more sites compared to the QD_SJN algorithm.

The DAVID system’s ability to integrate distributed heterogeneous databases and provide uniform access was demonstrated by its application to the Astrophysics Data System (ADS). The actual use of the proposed query processing algorithm for the DAVID prototype is illustrated in the context of the Astrophysics Data System.
Our experience with a sample of ADS databases has shown that our query processing algorithm reduced the processing cost over the previous semijoin based algorithm used by the DAVID system.

Future implementations of HDDBMSs will depend in part on how well they meet query processing performance requirements, particularly in terms of query response time. Furthermore, the successful implementation of query processing algorithms depends on the availability of accurate and efficient size estimation techniques. This work introduced a methodology for size estimations, tailored to the characteristics of very large astrophysics databases and applied to the Distributed Access View Integrated Database (DAVID) prototype which is an HDDBMS [8]. However, the basic ideas and computations given are generic enough to be easily transported to other DBMSs that use a global data model for heterogeneous data.

One of the primary objective throughout this work was the formulation of a methodology that would produce the most accurate size estimates possible. This could not be done without restricting its domain of application i.e., for databases where the majority of the queries are expressed on attributes with a small set of distinct attribute values. In the future, this methodology will be generalized by using a combination of parametric and non-parametric quantifiers to describe the data. This will help enhance the performance of the methodology when the underlying data have both the discrete and continuous characteristics.

A number of ongoing research efforts are underway at VPI and SU. For example, an alternative approach to the zero-one integer linear programming based formulation is being investigated. The possible alternative query processing approach
includes artificial intelligence based query tree search to obtain global optimal sequence of join operations. Further, extensions to database logic, to include heterogeneous database systems consisting not only of relational, hierarchical and network databases, but object-oriented and rule-based, are being investigated. One of the extensions involves the application of second-order logic.
References

1. Abiteboul, S., and Bidoit, N., "Non First Normal Form Relations to Represent Hierarchically Organized Data" in Proceedings of the third ACM SIGACT-SIGMOD symposium on Database systems, Waterloo, April 1984, pp. 191-200


47. Welch, J.N., "The Query Decomposition in Heterogeneous Databases" *M.S. Thesis, Dept. of computer science, University of Maryland*, College Park, Maryland


68. Bhasker, B., "An Implementation of the Query Processing Algorithm for the DAVID System" *Department of Computer Science, Virginia Polytechnic Institute and State University*, 1992
VITA

The author was born in Kanpur, India, in 1958. He received a B. E. degree in Electronics and Communications Engineering from University of Roorkee, India in 1980. In 1980, he joined the computer services division of the Tata Steel Ltd., India, as a software engineer. The author enrolled in the Virginia Polytechnic Institute and State University in 1984 and received his M. S. degree in Computer Science and Applications in 1986. The author joined Hughes STX Corporation in 1986, where he participated in a NASA sponsored research at the Goddard Space Flight Center, leading to the development of a heterogeneous distributed database management system called DAVID. The author took a leave of absence during 1987-1988 academic year and enrolled for the Ph. D. program at the Virginia Polytechnic Institute and State University. In 1988, he began the research on the query optimization in heterogeneous distributed database management systems. He will continue his research on the object oriented and heterogeneous database management systems at the Goddard Space Flight Center, Greenbelt in Maryland.

B. Prasad

224