Consideration of interdependencies in the relational database system, and, A proposal and evaluation of an expert system for the relational database structure

Timothy Jon Arndt
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
https://digitalscholarship.unlv.edu/rtds/158

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600
Consideration of interdependencies in the relational database system, and, A proposal and evaluation of an expert system for the relational database structure

Arndt, Timothy Jon, M.S.

University of Nevada, Las Vegas, 1991
Consideration of Interdependencies in the Relational Database System and
A Proposal and Evaluation of an Expert System for the Relational Database Structure

by

Timothy Arndt

A thesis submitted in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Science

Department of Computer Science
University of Nevada, Las Vegas

November 19, 1991
©1991 Timothy J. Arndt
All Rights Reserved
The thesis of Timothy Jon Arndt for the degree of Master of Science in Computer Science is approved.

Chairperson, Kia Makki Ph.D.

Examining Committee Member, Evangelos Yfantis Ph.D.

Examining Committee Member, Yonina Cooper Ph.D.

Graduate Faculty Representative, Jacqueline Brown Ph.D.

Graduate Dean, Ronald Smith Ph.D.

University of Nevada, Las Vegas
November 19, 1991
Abstract

This thesis addresses the issue of interdependencies in Distributed and non-Distributed Relational Database Management Systems and proposes a design and development of an Expert System that will hope to manage and enhance the current available Database Structures.

The Thesis consists of two parts. In the first part, we study, compare and evaluate the interdependencies found in the operating environment relevant to the Distributed Relational structure. Hardware and software configurations are grouped and compared in an attempt to understand the interdependencies of the system so that an optimal configuration may be obtained.

In the second part, we designed and developed an Expert System configuration with ease of use and functionality as foremost concerns. The design and configuration were also designed to support several platforms and query languages. The basic performance features factored into the system design are based upon data allocation and query frequency. The system reuses the transient tables used to service queries to achieve a performance improvement without explicit user knowledge. Basic fragmentation principles are also used to aid in performance by implicitly restructuring the tables within a database to balance access time. Also seen are several byproducts of the Expert System which, although may not have performance benefits will improve functionality of the current Relational Database System.
# Contents

1 Introduction 7

1.1 Introduction ................................................................. 7
1.2 Key Attributes of the Distributed RDBMS ......................... 8
1.3 Motivation ................................................................. 9
1.4 Related Work ............................................................ 11
   1.4.1 Hardware Configurations. ................................. 11
   1.4.2 Communications Network. .............................. 12
   1.4.3 Fault Tolerance............................................... 12
   1.4.4 Fragmentation Schemes. ................................. 13
   1.4.5 Query Semantics............................................. 13

2 Interdependencies, A Discussion and Analysis 14

2.1 The Five Phases of System Existence ............................. 14
2.2 The Six Classes of Hardware Configurations .................... 15
2.3 Ideal Distributed System Characteristics ........................ 17
2.4 Relational System Characteristics Classified .................. 19
2.5 RDBMS Attributes and Hardware Structures ................. 23
2.6 Operation Types and File Allocation Schemes. ................ 25
2.7 Data Allocation Strategies ........................................ 27
2.8 Rating of a class with specific attributes ...................... 30
2.9 Matrices of Configurations ......................................... 32
3 Expert System Proposal and Analysis

3.1 Expert System Component Interfaces .......................................................... 38
3.2 Expert System Proposed Enhancements .................................................... 39
3.3 Expert System's Key Processing Components........................................... 39
3.4 Pseudo Code of Expert System ................................................................. 43
  3.4.1 Transaction Manager ..................................................................... 43
  3.4.2 Initial Request Manager .................................................................. 45
  3.4.3 Log Manager ................................................................................... 48
  3.4.4 Semantic Query Optimizer ............................................................. 51
  3.4.5 Priority Request Manager ............................................................. 55
  3.4.6 System DB Status Manager ............................................................ 56
  3.4.7 Internal Queue Manager .................................................................. 57
  3.4.8 Internal Query Manager ................................................................. 58
  3.4.9 Reconstruction Manager .................................................................. 58
  3.4.10 Internodal Request Manager ......................................................... 59
3.5 Record Types for Expert System .............................................................. 60
3.6 Expert System Table Structures ............................................................... 67
3.7 Expert System Packeting ........................................................................... 70
  3.7.1 Packeting Structures ....................................................................... 70
  3.7.2 Packeting Transition Examples ....................................................... 77
  3.7.3 Creating a Table ............................................................................. 78
B.2 Single processor per node, multi-node. (CLASS 2) .................................. 122
B.3 Multi-processor, multi-node with $P > N, P < N \times 2$. (CLASS 3) .... 123
B.4 $P$ processors, and $N$ nodes where $P = N \times 2$. (CLASS 4) ........ 127
B.5 $P$ processors and $N$ nodes where $P > N, P = N \times 2 + I$. (CLASS 5) 131
B.6 $P$ processors and $N$ nodes, where $P > N$ and $P = N \times I$ where $I > 2$.
   (CLASS 6) ................................................................. 135

Glossary .............................................................................................................. 140
List of Figures

3.1 Expert System Component Interfaces ................................................... 38
3.2 Fragmentation Graphically Depicted ...................................................... 67
3.3 Create Packet Structure ................................................................. 71
3.4 Retrieve Packet Structure ................................................................. 72
3.5 Replace Packet Structure ..................................................................... 73
3.6 Append Packet Structure ..................................................................... 74
3.7 Delete Packet Structure ...................................................................... 75
3.8 Transition Packet Stages ...................................................................... 76
3.9 Create Packet example Part 1 .............................................................. 79
3.10 Create Packet example Part 2 .............................................................. 80
3.11 Retrieve Packet example ..................................................................... 81
3.12 Delete Packet example ....................................................................... 82
3.13 Creating a Database ........................................................................... 84
3.14 Loading the Table .............................................................................. 85
3.15 Creation of Transient Table ............................................................... 86
3.16 Creation of a second Transient Table .................................................. 87
3.17 Creating Transient Table using previous Transient Table .................... 88
3.18 Initial Table Structure prior to Local Fragmentation ............................ 89
3.19 Creation of Transient Table prior to Local Fragmentation ................... 90
3.20 Threshold Met, Local Fragmentation occurs ....................................... 91
3.21 Initial Table Structure prior to Remote Fragmentation ........................ 92
3.22 Creation of Transient Table on predicate, prior to Remote Fragmentation ................................................. 93
3.23 Threshold Met, Remote Fragmentation occurs ................................................................. 94
3.24 Initial Table Structures during Query-only state ............................................................. 95
3.25 Additional Transient Tables created from prior request ............................................... 96
3.26 Initial Table structures during Update state ..................................................................... 97
3.27 Fragmentation in the Update state .................................................................................... 98
3.28 Query service in remotely fragmented database ............................................................. 99
3.29 Transient Tables built to service remote request ........................................................... 100

4.1 Relational DB structure .................................................................................................. 103
Chapter 1

Introduction

1.1 Introduction

Achieving optimal performance through specific schemes for fragmentation, fault tolerance, and other areas is not a new topic and has resulted in several outstanding papers in recent years \(^1\). Yet, with numerous researchers working on each of the diverse topics of performance within the system the interdependent performance factors are generally not considered when justification and modeling are presented. Too often researchers and authors take a microscopic view when theorizing a performance benefit, which leads to a lack of interconnectability of all features of the distributed system\(^2\) to offer an optimal system configuration. Interdependency considerations need to be emphasized and incorporated into system development and performance planning.

The lack of definitive guidelines for designing schemes to support these interdependency relations has lead us to the research and discussion found in this paper. Although distributed systems by definition have several attributes in common, certain attributes such as hardware and communications platforms divide them into several classes which must be viewed differently.

Understanding the rudimentary characteristics of each of the supportive structures that make up the distributed system is important in understanding their interdependency within the system and is the key to development of an optimal system

\(^1\)A summary of our findings are given as references in the bibliography

\(^2\)Within this paper Distributed Systems refers to Distributed Relational Database Systems
After having concentrated an effort on compiling information relative to performance benefits, maintaining data integrity and retaining the needed emphasis on hardware restrictions, we found two important points to be most outstanding. First, no one configuration of hardware and allocation/retrieval schemes is optimal for all instances. Second, once an optimized configuration is attained, the dynamic nature of Relational Database Management Systems \(^4\) reduces the performance until redesign is inevitable. The first problem is discussed and an quantitative approach is derived in the second chapter. The second problem is reduced by the design and development of an Expert System\(^5\) configured to handle the problems induced by the dynamic and highly I/O driven natures of the RDBMS.

### 1.2 Key Attributes of the Distributed RDBMS

According to our investigation, five major attributes delegate the optimum performance of a distributed system. The hardware configuration is one major attribute that is usually a constant in any given distributed system and is often the focus of redesign to gain optimum performance in request serving. The communications network also represents a major attribute of the system and its vulnerability and instability handicap the system with costly fault tolerance schemes which have been devised to ensure data availability. Fragmentation schemes have been used to offset hardware and communication deficiencies to attempt to develop optimal systems. But as the use of fragmentation schemes has proliferated, the analysis of query semantics and query frequencies also become key factoring attributes in an optimal configuration.

This two part paper first analyzes the present construction of Distributed Systems and compares hardware and software configurations that delegate their performance.

In the second portion of the paper an enhancement is designed and developed to model an Expert System to return optimal performance while attempting to reducing system overhead and maintenance.

---

\(^3\)See chapter 4 for definition of Relational Databases and the Glossary for terms used in this paper

\(^4\)Relational Database Management Systems ... also referred to as RDBMS

\(^5\)In chapter 3 specifics of the design are given
1.3 Motivation

Ease of use and performance have always been the main motivators for change in the field of computer science. Most often the performance issue is considered first leaving the ease of use issue to a best fit philosophy afterwards. With the advent of Relational Database systems we see a somewhat reversal of priorities to attaining these two goals. Although Relational Databases offer the user phenomenal ah-hoc capabilities and a more English like presentation of data there are still several restrictions of the Relational Database Method. Please refer to chapter 4 for a further description of these restrictions and a short description of what a Relational Database System is if you are unfamiliar with this technology.

There are several excellent publications and methodologies to improving performance in the Relational Database System and even some that approach issues of ease of use, but to our knowledge there are no publications or methodologies that attempt to embrace both performance and ease. Even as rare as the combination of both is the occurrence of consideration of a performance benefits affect on all surrounding elements of the Relational Database, where the largest impacts are felt when processing in the distributed environment.

In attempts to understand both Relational Database Systems and Distributed Processing we found that several areas are being researched as possible avenues for performance benefits. There are several publications \(^6\) that offer valid and exciting new approaches to current technological inadequacies, but none consider the interdependencies of the several entities of the Distributed Relational Database System. Such issues as data integrity, node failure, and coprocessing have the tendency to highly affect the performance of each of the other entities.

In this two part thesis we identify the interdependencies within the Distributed Relational Database System and offer two approaches at attempting optimal performance without sacrificing ease of use. The first approach uses the identification of these interdependencies and a matrix to offer hardware configurations that attempt to return optimal performance. The second approach is the development of an Expert System that evaluates the interdependencies and dynamically reconstructs entities within the system to obtain optimal performance without intervention or affecting user applications.

\(^6\)Refer to bibliography
To really understand how the proposed Expert System would effect the commercial Relational Database marketplace we have included here several quotes from two of the most widely referenced sources of information in the IBM mainframe environment. These exact quotes reflect the lack of tools for the management and analysis of the Relational Database Systems.

...But in spite of such success, and the fact that DB2's7 performance has improved more than tenfold since its introduction and is constantly improving, the DB2 performance issue is very real...

...Traditionally, performance problems are solved in one of two ways: buy bigger boxes on which to run the applications or tune the systems...

...While tuning certainly is useful, in case after case, it becomes clear that the real, long-term foundation for performance in the DB2 environment is system design[Davydov 91]...

...There is high human resource cost in controlling and administering each DB2 subsystem...

...Also, an enormous amount of DBA activity is required to manage multiple subsystems[Werman 91]...

...DBA's systems programmers and management are juggling responsibilities as they attempt to control the DB2 environment in their respective shops. There is a crisis brewing...

...The first high-volume transaction applications in DB2 have another problem; They do not use relational techniques. Instead, they merely use DB2 as an access method for older-style applications...

...Most shops are not ready for large-scale DB2 production. There are cultural, political and staffing issues as well as managements failure to understand the problems involved in transition to a DB2 production world...

...Moving into a more efficient structured organization to support the use of DB2 allows selection of a tool set to support operational needs. Select tools based on the organization rather than structuring the organization based on the tools. One of the BIGGEST problems in the DB2 tool market is still evolving and there is a lack of integrated tools to fulfill every need[Werman 90]...

---

7IBM's SQL based Relational database System
...Its purpose is to disclose some performance tradeoffs and in particular to show some areas where the query optimizer is arguably deficient...

...The first usually alludes to the fact that if the rows are too long for 4096-byte pages, the page size goes to 32768[Snyder 89]...

1.4 Related Work

The discussion of interdependencies within the Distributed Relational Database System is not a widely published topic and as such we have no direct previous work to refer to or compare against. The development of database Expert Systems can be seen materializing as byproducts of the latest assistance based monitoring tools, but no direct correlation between performance and dynamic restructuring is seen either as a proposal or commercially available product.

In considering the interdependencies on which the second chapter of this paper is based, several of the better articles published are mentioned and highlights of their contributions summarized. We list several publications in the bibliography which are key readings. The reader is invited to read at least one of the publications from each group such that a deeper appreciation of this paper’s contributions may be gained.

There are five general areas in which system performance schemes can be grouped. These five areas are hardware configuration, communications schemes, fault tolerance schemes, fragmentation schemes and query semantics.

To avoid personal preference and condense the seemingly endless collection of excellent work in each of the areas, we will only highlight material from a select few authors under each category.

1.4.1 Hardware Configurations.

In 1991, Kia Makki and Timothy Arndt published a discussion of Distributed Relational Database Interdependencies, creating matrices\(^8\) for comparison between configurations in attempts to obtain optimal performance through consideration of interdependencies in the RDBMS.

\(^8\)Given as support material in chapter 2.9 and appendix B
In 1990, Pirahesh and Mohan discussed Parallelism in Relational Database Systems covering architectural issues and design approaches. The paper covered the pros and cons of the shared nothing, shared disk, and shared everything architectures. The authors also discussed the performance implications of parallelizing complex queries in the distributed system.

1.4.2 Communications Network.

Most publications in this area focus on the need for fault tolerance schemes and in the event of parallel or coprocessing, the problem of concurrency control mechanisms.

In 1989, J. Glasgow and G. MacEwen presented a paper on the Operator Net approach to modeling distributed systems. Operator Nets, first introduced by Ashcroft and Jagannathan in 1985, are graphical languages which provides a method for describing interprocess communications and parallelism in a distributed computing environment.

In 1986, Leslie Lamport discussed the mutual exclusion problem and several solutions. This was an outstanding paper that covered several aspects of the mutual exclusion problem.

1.4.3 Fault Tolerance.

In 1989, A. Borg, W. Blau, W. Graetsch, F. Hermann and W. Oberle discussed the use of three way message transmission to enhance fault tolerance in the UNIX communications subsystem. The authors used a backup process pathing that allows for transmission and process failure without message failure.

In 1987, G. Bracha and S. Toueg offered a scheme for enhancing fault tolerance in the distributed system by offering a deadlock detection scheme.

In 1987, F. Mattern discussed the distributed system problem of knowledge by all nodes of the global system state. The author used a asynchronous message passing process instead of the traditional synchronous method.
In 1985, H. Garcia-Molina and D. Barbara used assigning votes in the distributed system to manage mutual exclusion. The assignment of votes enforces mutual exclusion without enforcing communications between groups. The authors discussed situations and solutions related to node and communications subsystems failure. Limitations and performance of their proposals are also discussed.

1.4.4 Fragmentation Schemes.

In 1990, B. Gavish and O. Sheng discussed the level of file replication and allocation necessary to achieve a satisfactory level of system performance. Replication and dynamic file migration are discussed as acceptable alternatives. The authors used adaptive and nonadaptive models as comparisons for efficiency.

In 1988, P. Apers discussed the optimization of distributed database systems via fragmentation principles. The paper also discussed the cost of data allocations, the forking processes and forking graphs, and the computation of optimal allocations given static schedules.

In 1985, D. Sacca and G. Wiederhold discussed partitioning and allocation as a critical aspect of the distributed database design effort.

1.4.5 Query Semantics.

In 1988, P. Valduriez and S. Khoshafian discussed the highly suitable situation of servicing a complex query in a fragmented relational database for parallelism. Phenomenal performance benefits were returned in the order of one and two magnitudes.

In 1988, A. Farrag and M. Otsu offered a proposal to increase concurrency by using knowledge of the physical fragmentation of the data and semantics of the query. The process is based on nonserialized events to increase throughput by relieving interleaving events.
Chapter 2

Interdependencies, A Discussion and Analysis

2.1 The Five Phases of System Existence

Although the life span of a particular system is generally 5 to 8 years and is the accepted guideline in financial justification, it can often times be less than 2 years. The key to justification is in foresight of design flexibility, realistic evaluation of the end users needs, and above all the ability to increase functionality of the system which it replaces.

- The Five Phases of Existence.
  1. The Analysis Phase
  2. The Justification Phase
  3. The Design Phase
  4. The Implementation Phase
  5. The Maintenance/Enhancement Phase

- Analysis Phase: Encompasses the need for evaluation of query types, query semantics, recovery needs, frequencies of queries and acceptable end user response time.

- Justification Phase: Encompasses the need of the system based upon growth rate, unacceptable response time and financial considerations.
Design Phase: Uses output from analysis phase to create a suitable system based upon software and hardware available.

Implementation Phase: Consuming the most resources and cost, the implementation phase consists of applying a given programming language and data access methods to solve the process defined in the analysis phase. Although the most expensive phase to undergo, it should be relatively simple to complete.

Maintenance Phase: Consideration for query types, query frequencies, recovery needs and hardware configurations changes are major concerns during the maintenance phase. When changes to present system become too costly to implement, or hardware advancements offer system improvements beyond what the present system can offer, a contingency analysis and justification should be done to possibly replace or enhance present system.

As can be seen, it is critical to system life expectancy to design not only for present needs but also short term future needs. Consideration of all aspects of the system can be an overwhelming task if done simultaneously but if a scheme is used to add atomicity to the design, allowing for detailed considerations individually when possible, and will aid in design simplicity.

It is seen that data allocation is the only area of design that shows interdependencies with all system functions, and will be the focus of this paper.

2.2 The Six Classes of Hardware Configurations

Class One is the data server configuration and the mother of distributed computing. This configuration is simple in that data is concentrated in one location, thus processing is simplified. The obvious drawbacks include data availability in case that node is removed and distribution of data retrieval, among others.

Class Two includes configurations where there is a single processor for each node. This is probably the most common configuration and easiest to implement using distributed allocations. Often, this scheme will introduce features that will offer an optimal configuration. There is one minor inadequacy with this configuration though, the fact that node fault tolerance is not optimal. An example: if a node is removed, data availability and possible data integrity is in question. A proper logging scheme could help to insure data integrity and eventual data availability by reproduction of
the data at another node, but reproduction is often time consuming and normally unnecessary.

*Class Three* includes systems that offer more processors than nodes, but not enough extra processors that each node could have at least two processors. This class is a slight variation of class two except that one or more nodes could offer co-processing at certain nodes. Although this type of configuration could offer more optimal performance than class two by coprocessing at certain nodes, it introduces complication of handling communications between the nodes which requires different operating system software at each node containing a different processor configuration. Data allocation and fault tolerance schemes also become more costly, generally offsetting the savings by increased overhead.

*Class Four* includes processor node groupings where there are exactly two processors per node. This configuration is unique in that it offers all the advantages of class two groupings and covers their shortcomings without over complicating the total system. Such configurations can support processor connections where the processors are either loosely coupled or tightly coupled, each has its distinct advantages. This class of configurations promises to display the most optimal performance when combined with other features of the distributed system. The reason for this is that whether coupled loosely or tightly they offer all characteristics of class two configurations along with improving data availability and node fault tolerance by introducing local and immediate recovery if a processor should fail. Loosely coupled configurations lack the coprocessing capabilities of the tightly coupled system, but in data versus computational intensive systems, loosely coupled systems offer simplified data retrieval locking mechanisms and lend themselves to simplified data recovery.

*Class Five* groupings are node processor groupings where \( M = N \times 2 + I \) where \( M \) is the number of processors, \( N \) is the number of nodes and \( I \) is some integer. These configurations are a variation of class three processors, and although they can offer performance enhancements similar to class four groupings, they still incur operating system overhead similar to class three groupings which will degrade performance in data intensive systems below that obtainable by a class four configuration.
Class Six groupings are node processor configurations where \( M = N \times I \), \( M \) being the number of processors, \( N \) being the number of nodes and \( I \) being some integer. These configurations are somewhat a variation of the class three groupings in that operating system software would be the same for every node. Systems of this class lend themselves to tight coupling and computational intensive applications. Although processor speedup is evident, data retrieval and data integrity induce an obvious overhead, and in return offer less than linear speedup, especially when involving node recovery.

Having generalized hardware configurations, it is necessary to define other attributes of the system that will affect performance, and can be permuted through each class. Since the most optimal classes offer multiple processors per node, the topic of shared or non-shared data is important. Managing shared data can become costly in retrieval and guaranteeing integrity, but non-shared data removes ability for instantaneous local node recovery. The best fit of both configurations would somehow include a facility that supports shared data but disallows it except for recovery or data reallocation.

The topic of performance related to locality or computation becomes an important factor. Although local computation decreases the complexity of the computation, it reduces optimal computational performance by not using the maximum allowable processors. Again, an acceptable means of mixing the computational locality will prove to be most optimal in general, but not guaranteed for all configurations. The additional permutations of processor couplings at nodes can modify a class's ability to perform optimally. Pros and cons are defined in each class's generalization.

2.3 Ideal Distributed System Characteristics

At the heart of any distributed system are certain requirements that must be met regardless of implementation. These include:

1. A query processing scheme that performs optimally for hardware structure implemented.

2. Fault tolerance mechanisms for communication links and processors.

3. A concurrency control mechanism that performs with low overhead to total system operation.
4. Flexibility on database design, assignment of locality of data should be decided by the operating system not the user. In addition, the allocation algorithm should be able to optimally handle allocations in a dynamic environment. The algorithm must work well with single attribute assignments or complete tuple assignments.

5. At least minimal monitoring capability, this will allow for ease of incorporating load balancing and system tuning and performance evaluation.

6. Scheme for implementation should not attempt parallelism of computation if single processor groupings in a geographically distributed system. This technique only lends itself to closely coupled processor groupings.

7. A log must be established, this serves many real life purposes, and will prove to aid in concurrency and backout control. Tracing of transaction execution not only must be auditable but will not cost much in processor or I/O overhead. Logging is useful in 'Change Accumulation' backout processing, monitoring of system performance, and data recreation upon full node failure.

8. Weighting system for queries. Some types of queries will lend themselves to certain allocation and execution structures that will degrade performance of other queries.

9. Data availability, and reduced down time. The allocation structure must be defined so that no loss of data is possible, or data inaccessibility will result.

10. Allocation structure and concurrency control must be allowed to achieve maximal growth potential. Black Box additions to existing system structure must be possible. Reconstruction of data for a lost node is possible from the transaction log and original data established at some checkpoint in processing.

11. Locality of query request should not degrade query serviceability. That is, any query should be acceptable from any node in the system without response time degradation.

12. Methods for efficiently handling data retrieval when a node becomes unavailable must also be exist.

13. Development of the system should be with view of semioptimal performance with transparency to worst case. Too often algorithms are designed around
worst case, degrading performance of the system to support this scenario even when worst case may occupy less than 1 percent of total processing.

14. Data allocations must not be site committed. Committed allocations set the stage for impossible fault tolerance.

15. The system must use semantic and rule structures of query operations to generate data allocations and hardware configurations. Do not design the query or allocations around the hardware. This will allow for higher portability of the system, along with other obvious reasons.


2.4 Relational System Characteristics Classified

System configurations are given to break each of the six classes into their associated single system examples with the different permutations of attributes previously defined. These configuration types will be used to weigh each of the systems on their potential to return optimal performance with each of the permutations. The designer should be able to take this matrix and determine the optimal configuration for supporting his or her system by deciding which attributes of the system when permutated will return the highest degree of optimum performance.

Each of the six classes of systems must be compared relative to the necessary attributes of a distributed system. Following are the fifteen relative features of a distributed system and those attributes that when permutated result in a rating for a specific class and permutation. All classes and configurations are rated 0 to 9, 0 being the lowest and 9 being the highest. Each rating is relative to all other classes and configurations. What-if combinations of attributes can lead the reader to establishing a comparison base of his or her present system with another possible permutation of attributes.
Characteristics and Attributes of the Distributed Relational System that affect these characteristics.

1. Data Availability:
   - Number of processors per node.
   - Use of shared, non-shared or minimally shared storage.
   - If multiple processors per node, if tightly or loosely coupled.
   - If communications between nodes is node to node, multiple path token ring or minimal path token ring.

2. Processor Speedup:
   - Number of processors per node.
   - If queries are serviced by host node only or host and requested nodes using semantic minimization.

3. Design Flexibility:
   - Use of shared, non-shared or minimally shared storage.
   - If multiple processors per node, if tightly or loosely coupled.
   - If communications between nodes is node to node, multiple path token ring or minimal path token ring.

4. CPU Fault Tolerance:
   - Number of processors per node.
   - Use of shared, non-shared or minimally shared storage.
   - If communications between nodes is node to node, multiple path token ring or minimal path token ring.

5. Communications Fault Tolerance:
   - If communications between nodes is node to node, multiple path token ring or minimal path token ring.

6. Support Logging:
   - If multiple processors per node, if tightly or loosely coupled.
If communications between nodes is node to node, multiple path token ring or minimal path token ring.

If queries are serviced by host node only or host and requested nodes using semantic minimization.

7. Recoverability:

Use of shared, non-shared or minimally shared storage.

If communications between nodes is node to node, multiple path token ring or minimal path token ring.

8. Support Query Optimization:

Use of shared, non-shared or minimally shared storage.

If multiple processors per node, if tightly or loosely coupled.

If queries are serviced by host node only or host and requested nodes using semantic minimization.

9. Dynamic Restructuring:

Number of processors per node.

Use of shared, non-shared or minimally shared storage.

If multiple processors per node, if tightly or loosely coupled.

If communications between nodes is node to node, multiple path token ring or minimal path token ring.

10. Support Monitoring:

If multiple processors per node, if tightly or loosely coupled.

If communications between nodes is node to node, multiple path token ring or minimal path token ring.

11. Query Weighting:

If multiple processors per node, if tightly or loosely coupled.

If queries are serviced by host node only or host and requested nodes using semantic minimization.
12. Data Locality:

Use of shared, non-shared or minimally shared storage.
If multiple processors per node, if tightly or loosely coupled.

13. Data Commitment:

Number of processors per node.
Use of shared, non-shared or minimally shared storage.
If multiple processors per node, if tightly or loosely coupled.

14. Support Semantic Query:

Use of shared, non-shared or minimally shared storage.
If queries are serviced by host node only or host and requested nodes using semantic minimization.

15. Total Time Speedup:

Number of processors per node.
Use of shared, non-shared or minimally shared storage.
If multiple processors per node, if tightly or loosely coupled.
If communications between nodes is node to node, multiple path token ring or minimal path token ring.
If queries are serviced by host node only or host and requested nodes using semantic minimization.

Each of these attributes of the distributed system affect total overall performance of the system with some affecting the system more than others. Critical characteristics include data availability, design flexibility, CPU fault tolerance, communications fault tolerance, extent of supporting query optimization, the locality of data storage and its extent of semantic query processing.
2.5 RDBMS Attributes and Hardware Structures

To be able to evaluate fairly the different hardware configurations, and data allocations of different systems it is necessary to build a matrix of the systems versus the necessary and desired attributes.

This matrix will help to clarify how certain physical attributes of the distributed system in the following 17 permutations compare with one another on critical characteristics.

Assuming parallel system types, distributed geographically.

1. Single node with N processors serving M sites. Sites served through communications links. Shared storage, processors tightly coupled.

2. Single node with N processors serving M sites. Sites served through communications links. Data distributed in share-nothing format. Processors are tightly coupled.


5. Single node with N processors serving M sites. Sites served through communications links. Data distributed in single update, multi-read format. Processors are loosely coupled.


7. Single node with N processors serving M sites. Sites served through communications links. Data distributed in multi-read, multi-write format. Processors are loosely coupled.
8. M nodes with M processors. Sites served through communications with local computations. Data distributed in share-nothing format. Data retrieved at remote site only.

9. M nodes with M processors. Sites served through communications with local computations. Data distributed in share-nothing format. Subquery computation at local site.

10. M nodes with N processors. \( N > M \) evenly where \( pM = Np = 2,3,4,5,... \). Communication through links with local computation. Share-nothing data format. Processors are loosely coupled.

11. M nodes with N processors. \( N > M \) evenly where \( pM = Np = 2,3,4,5,... \). Communication through links. Specific number of processors are used as data servers. Data is share-nothing format. Processors are loosely coupled.

12. M nodes with N processors. \( N > M \) evenly where \( pM = Np = 2,3,4,5,... \). Communication through links. Specific number of processors are used as data servers. Data is in multi-read single write format. Processors are loosely coupled.

13. M nodes with N processors. \( N > M \) evenly where \( pM = Np = 2,3,4,5,... \). Communication through links. Specific number of processors are used as data servers. Data is in multi-read multi-write format. Processors are loosely coupled.

14. M nodes with N processors. \( N > M \) evenly where \( pM = Np = 2,3,4,5,... \). Communication through links. All processors used in data retrieval. Data is stored in share-nothing format. Processors are loosely coupled.

15. M nodes with N processors. \( N > M \) evenly where \( pM = Np = 2,3,4,5,... \). Communication through links. All processors used in data retrieval. Data is stored in share-nothing format. Processors are tightly coupled.

16. M nodes with N processors. \( N > M \) evenly where \( pM = Np = 2,3,4,5,... \). Communication through links. All processors used in data retrieval. Data is shared in a multi-read single write format. Processors are tightly coupled.

17. M nodes with N processors. \( N > M \) evenly where \( pM = Np = 2,3,4,5,... \). Communication through links. All processors used in data retrieval. Data is shared in a multi-read multi-write format. Processors are tightly coupled.
2.6 Operation Types and File Allocation Schemes.

There are eight distinct operations in the Relational Database System:

- Select.
- Project.
- Join.
- Natural-Join.
- Union.
- Cartesian Product.
- Set Difference.
- Intersection.

The basic operations are select, project, cartesian product, union and set difference, all others can be built from these.

File allocation strategies widely range from single file allocation to vertical/horizontal fragmentation. The following is a description of each type of allocation and comparisons of supportive structures of each.
1. Single processor, M nodes in a single file structure.

2. Multi-processor, M nodes in a single file structure.


4. M processors, M nodes with one file per node. Whole file fragmented horizontally.

5. M processors, M nodes with one file per node. Whole file fragmented vertically.

6. M processors, M nodes with one file per node. Whole file fragmented vertically / horizontally.

7. N processors, M nodes, with \( N = M^* (2,3,4,5.....) \). One file per node with the data fragmented horizontally.

8. N processors, M nodes, with \( N = M^* (2,3,4,5.....) \). One file per node with the data fragmented vertically.

9. N processors, M nodes, with \( N = M^* (2,3,4,5.....) \). One file per node with the data fragmented vertically and horizontally.

10. N processors, M nodes, with \( N = M^* (2,3,4,5.....) \). One file per node with the data available only to its assigned node.

11. N processors, M nodes, with \( N = M^* (2,3,4,5.....) \). One file per node with the data available only to its assigned node. Data is fragmented horizontally.

12. N processors, M nodes, with \( N = M^* (2,3,4,5.....) \). One file per node with the data available only to its assigned node. Data is fragmented vertically.

13. N processors, M nodes, with \( N = M^* (2,3,4,5.....) \). One file per node with the data available only to its assigned node. Data is fragmented vertically and horizontally.

The following is a matrix which can be used to show the performance that could be expected from each of the actions under the relative hardware structures.
2.7 Data Allocation Strategies.

From combinations of the hardware configurations and query types we can construct an optimal system configuration. For the most part, query types of retrieval by projection favor a multi-processor, vertically fragmented files, distributed using semantics and query frequencies to define allocations.

Query types that reflect inserts show similar favoritism to that of retrieval. Query types that reflect updates will favor horizontal fragmentation and distributions using semantics.

Although horizontal fragmentation and data locality using semantics seem to be the favored allocation scheme, consider this. If in a highly dynamic system group updates are done often against specific attributes of a tuple, a simple division of the vertical fragmentation on that attribute type into two separate horizontal fragmentations could reduce the query, update and insert process by as much as half the original. At worst, the binary search time for a specific query would be reduced. It is further obvious that at worst case the binary search for a specific attribute will result in at least a 50 percent reduction in search time for the tuple. Futher splitting could linearly reduce the search time. Caution must be exercised in the extent of this horizontal splitting, for eventually the tradeoff gained by splitting the vertical fragments further will be offset by the query cost induced by calculating which fragment to search. Fragment storage costs will also influence splitting. Probably the best approach that would offer advantages of both vertical and horizontal fragmentation would be to physically fragment the data vertically and create an index on the fragment with a restricted amount of entries that would give offsets to a few select tuples in the fragment. An index with restricted sizing would show optimality at minimal
sizes, and maintenance of the index would be minute.

The real key to data allocation does not lie in such simple schemes as this, although they will augment the overall efficiency. The real impact to data allocation schemes is seen in layout requirements of minimal communication and storage.

The use of semantics, anticipatory query types, and frequencies in deciding fragmentation strategies is unchallenged when it comes to deciding a basic fragmentation strategy. The only aspect of fragmentation that still raises doubt is the question of efficiency gained by distributing the data among the several nodes of the database. Each approach to distribution has its highlights, but the overall query frequencies will always delegate which method will perform optimally. Several authors try to increase the optimality of systems by allowing data migration or data reallocation as the frequencies of queries change. We do not agree with this strategies. If a system is designed to support this type of processing, it will undoubtedly contain a higher overhead and unless ad-hoc queries abound in the system, the cost of the overhead will drastically outweigh benefits gained by allowing for migration. Understand though, that data reallocation must be supported in a system that will offer acceptable data availability, and without it, fault tolerance at node level cannot be established.

Hardware configurations, that include network and node complex structures also play a significant role in establishing data allocation strategies. In discussing configuration alternatives we assume a few basic features of nodes and the communications network, without giving details. First, processor failures are acceptable and anticipated, either on a permanent or temporary basis. Second, each node is assumed to have a limited Uninterruptable Power Supply\(^1\) capable of maintaining power long enough to transfer locally held fragments to alternate sites if a lengthy outage seems evident. Third, a communication network is in place that will support \(N^2\) paths to every other node. \(N\) will be decided upon the volatile nature of the network. And fourth, that a working logging/synchpoint mechanism is in place over the communications network that will support re-establishment of data integrity and availability if either the UPS or network should fail.

In the comparison of configurations, significant improvements in fault tolerance, speedup, and reliability are seen when multiple processor complexes are used. Communication networks become an overly complicated topic and we will avoid their definite interaction right now. Again, semantics and frequencies of query types dele-

---

\(^1\)also referred to as UPS

\(^2\)\(N > 1\)
gate structure, at each node, but coupling of the processors should require information based on processor reliability as to the internal structure of a node. Loosely coupled dyadic processors would prove to be the most optimal internal structure. We have a preference for dyadic versus N-processor groups and the constraint that they be loosely coupled for good reason. When processors are tightly coupled, the optimal use of processor power distribution must be delegated through code specifically designed to perform in a parallel processing environment, which introduces complexity into the system. There has been a considerable amount of progress in the parallel processing realm, but because of the need for specific coding structures, complexity of structure, the lack of both commercially available complexes, the lack of properly trained programming staffs and tightly coupled parallel processing machines need more refinement to make the parallel process transparent to the programmer before they will become commercially accepted. Dyadic processors, on the other hand, will allow for limited parallel processing at higher levels within the system, if found to be feasible.

The dyadic processors should also allow for shared data on a demand basis. By demand, we mean that data can be considered shared but data will be read only from one processor while the other processor will maintain update capability. When a processor fails, a request to Function Ship requests to the downed processor complex can be routed to its twin. At the time the processor fails, several actions can be taken, either a decision can be made to continue processing at the node with the processor that is still online by function shipping or, if the local system appears unstable, the twin processor can begin backout of current transactions against both processors and migrate data allocated locally to other nodes. The latter would induce tremendous overhead by request of data allocations throughout the system, along with network loading to distribute data and inform other nodes of data reallocation. The network at the disbanding site is assumed operational; this would include network controllers and the establishment of communication network connections to a remote processor. Although the network reconnection sounds like an expensive venture, in reality it requires minimal effort, if the network configurations are transparent to the local controllers.

If the node complex takes a complete hit, data availability will be degraded but data integrity will not be lost if a proper logging scheme is used. With the proper logging scheme data at the lost site can be reconstructed and reallocated at a new site. The degradation arises in the system from data being unavailable until all data is reconstructed by building up from a shadow file and change accumulation processing.
against the log. Synchronization point processing becomes an important aspect here along with a proper notification process.

As you can see, there are several necessary processes which must be maintained in an acceptable distributed parallel processing environment, but there are also those pieces which rely on variables induced by query type and frequency levels which delegate important system strategies.

In the third chapter we use the proposed Expert System to represent the complete structure of a defined distributed parallel processing system, including algorithms that support data allocation, fault tolerance, data availability and node complex internal workings in response to a query.

2.8 Rating of a class with specific attributes.

To obtain a clearer understanding of the meaning of these ratings, we give the following example that explains the reasoning behind the values given to each permutation.

Take class and configuration number 61. This system configuration places the system in class 4. Queries are serviced through the host query node only, meaning that if data needed to satisfy the query is also stored at other nodes in the system, only data retrieval with no semantic minimization will be done to aid in data transfer compression to lower query costs. Communications are by a token ring base communication using a minimized path algorithm. The processors are loosely coupled and multiple processors at the specific node do not share storage.

Comparing on the given rating scale the CPU fault tolerance of this configuration we must consider the effects of the number of processors at each node, the format of the data storage and the communication base. Since there are exactly two processors at each node, the system will rate higher than another configuration that has only one, or systems that do not consistently have the same number of processors at each node. The reason being is that a node containing only one processor per node will provide a lower fault tolerance per CPU compared with a system with multiple CPUs. The system with inconsistent numbers of processors at a node will lend itself to several different versions of operating systems at nodes in the system and must be designed to communicate with each other. A system with a consistent number of processors

\[^3\text{from Appendix B}\]
\[^4\text{P processors and N nodes where } P = N \times 2\]
at each node requires only one version of the supporting operating system. Systems
that contain higher multiples of processors at each node$^5$ may produce a higher total
speedup and other improvements.

Although storage format of data does not directly affect CPU fault tolerance,
the format does reflect interdependency with other characteristics$^6$ and must be con­
sidered. Having a nonshared storage format, the system will lack support of data
availability and semantic query processing.

The communications base used could be considered optimal as token ring offers
integrity and reduction of the I/O processing over node to node multiple path token
ring.

By combining the individual ratings based on the comparison of these three at­
tributes relative to all other configurations of all classes, a rating is given for CPU
fault tolerance for this configuration equal to six. The main reasons the configuration
did not receive an optimal rating of a nine are; that queries where serviced only by
the host, thereby not using resources of query optimization and not using semantic
minimization and thus servicing partial queries at remote nodes. The use of non­
shared data storage also degrades performance in that if a processor was disabled,
the data stored under its authority would become presently unavailable until it could
be reconstructed at another node.

Major contributors to optimal performance are the use of multiple processors at
each node to improve node availability and the use of minimized token ring com­
munications to insure the highest possible communication available with minimal
overhead.

---

$^5 P = N \times I$ where $I > 2$

$^6$like data availability
2.9 Matrices of Configurations

The following matrices represent our expected performance of each hardware configuration relative to the 15 attributes necessary in a distributed system. A description of each of the configurations can be seen in Appendix B. The rating values are in order from 0 for lowest to 9 being highest. The rating value given for each permutation is in relation to all other 161 permutations.

2.9.1 Class 1

<table>
<thead>
<tr>
<th>Attribute</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Data Availability</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>* Design Flexibility</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>* Fault Tolerance CPU</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>* Fault Tolerance Comm.</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Recoverability</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>* Support Query Optimiz.</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Support Dynamic Restruct</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>* Data locality Optimal</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>* Support Semantic Query</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>TOTALS of critical only</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>0</td>
<td>10</td>
<td>12</td>
<td>9</td>
<td>6</td>
<td>14</td>
<td>11</td>
<td>8</td>
<td>5</td>
</tr>
</tbody>
</table>

* indicated critical characteristics
configuration types are listed by number in Appendix B
comparison ratings are relative to all classes

2.9.2 Class 2

<table>
<thead>
<tr>
<th>Attribute</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Data Availability</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>* Design Flexibility</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>* Fault Tolerance CPU</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>* Fault Tolerance Comm.</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Recoverability</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>* Support Query Optimiz.</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Support Dynamic Restruct</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>* Data locality Optimal</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>* Support Semantic Query</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>TOTALS of critical only</td>
<td>26</td>
<td>26</td>
<td>41</td>
<td>30</td>
<td>42</td>
<td>40</td>
</tr>
</tbody>
</table>

* indicated critical characteristics
configuration types are listed by number in Appendix B
comparison ratings are relative to all classes
### 2.9.3 Class 3

<table>
<thead>
<tr>
<th>Attribute</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
<th>24</th>
<th>25</th>
<th>26</th>
<th>27</th>
<th>28</th>
<th>29</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Availability</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Design Flexibility</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Fault Tolerance Comm.</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Recoverability</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Support Query Optim.</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Support Dynamic Restruct.</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Data locality Optimal</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Support Semantic Query</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>TOTALS of crits</td>
<td>22</td>
<td>30</td>
<td>32</td>
<td>33</td>
<td>35</td>
<td>32</td>
<td>43</td>
<td>45</td>
<td>33</td>
<td>46</td>
<td>48</td>
<td>48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>31</th>
<th>32</th>
<th>33</th>
<th>34</th>
<th>35</th>
<th>36</th>
<th>37</th>
<th>38</th>
<th>39</th>
<th>40</th>
<th>41</th>
<th>42</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Availability</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Design Flexibility</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>8</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Fault Tolerance Comm.</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Recoverability</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Support Query Optim.</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Support Dynamic Restruct.</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Data locality Optimal</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Support Semantic Query</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>TOTALS of crits</td>
<td>34</td>
<td>46</td>
<td>48</td>
<td>35</td>
<td>49</td>
<td>51</td>
<td>35</td>
<td>33</td>
<td>35</td>
<td>36</td>
<td>36</td>
<td>38</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>43</th>
<th>44</th>
<th>45</th>
<th>46</th>
<th>47</th>
<th>48</th>
<th>49</th>
<th>50</th>
<th>51</th>
<th>52</th>
<th>53</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Availability</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Design Flexibility</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Fault Tolerance Comm.</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Recoverability</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Support Query Optim.</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Support Dynamic Restruct.</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Data locality Optimal</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>0</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Support Semantic Query</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>TOTALS of crits</td>
<td>35</td>
<td>46</td>
<td>48</td>
<td>36</td>
<td>49</td>
<td>51</td>
<td>37</td>
<td>40</td>
<td>51</td>
<td>38</td>
<td>52</td>
<td>54</td>
</tr>
</tbody>
</table>

* indicated critical characteristics
configuration types are listed by number in Appendix B
comparison ratings are relative to all classes

---

**33**
### 2.9.4 Class 4

<table>
<thead>
<tr>
<th>Attribute</th>
<th>55</th>
<th>56</th>
<th>57</th>
<th>58</th>
<th>59</th>
<th>60</th>
<th>61</th>
<th>62</th>
<th>63</th>
<th>64</th>
<th>65</th>
<th>66</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Data Availability</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>* Design Flexibility</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>* Fault Tolerance Comm.</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Recoverability</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>* Support Query Optim.</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Support Dynarm Restruct</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>9</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>* Data locality Optimal</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>0</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>* Support Semantic Query</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td><strong>TOTALS of criticals</strong></td>
<td>27</td>
<td>36</td>
<td>34</td>
<td>29</td>
<td>35</td>
<td>39</td>
<td>39</td>
<td>49</td>
<td>51</td>
<td>40</td>
<td>50</td>
<td>52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>67</th>
<th>68</th>
<th>69</th>
<th>70</th>
<th>71</th>
<th>72</th>
<th>73</th>
<th>74</th>
<th>75</th>
<th>76</th>
<th>77</th>
<th>78</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Data Availability</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>4</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>* Design Flexibility</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>* Fault Tolerance Comm.</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Recoverability</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>* Support Query Optim.</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Support Dynamic Restruct</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>* Data locality Optimal</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>* Support Semantic Query</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td><strong>TOTALS of criticals</strong></td>
<td>42</td>
<td>52</td>
<td>54</td>
<td>43</td>
<td>53</td>
<td>54</td>
<td>35</td>
<td>44</td>
<td>45</td>
<td>37</td>
<td>48</td>
<td>47</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>70</th>
<th>80</th>
<th>81</th>
<th>82</th>
<th>83</th>
<th>84</th>
<th>85</th>
<th>86</th>
<th>87</th>
<th>88</th>
<th>89</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td>* Data Availability</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>* Design Flexibility</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>* Fault Tolerance Comm.</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Recoverability</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>* Support Query Optim.</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Support Dynamic Restruct</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>* Data locality Optimal</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>4</td>
<td>5</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>* Support Semantic Query</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td><strong>TOTALS of criticals</strong></td>
<td>41</td>
<td>56</td>
<td>56</td>
<td>45</td>
<td>56</td>
<td>58</td>
<td>45</td>
<td>58</td>
<td>63</td>
<td>47</td>
<td>57</td>
<td>56</td>
</tr>
</tbody>
</table>

* indicated critical characteristics
configuration types are listed by number in Appendix B
comparison ratings are relative to all classes
### 2.9.5 Class 5

<table>
<thead>
<tr>
<th>Attribute</th>
<th>01</th>
<th>02</th>
<th>03</th>
<th>04</th>
<th>05</th>
<th>06</th>
<th>07</th>
<th>08</th>
<th>09</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Availability</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>Design Flexibility</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>Fault Tolerance Comm.</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>Recoveryability</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>Support Query Optimism</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>Support Query Weighing</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>Data locality Optimal</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>Support Semantics Query</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td><strong>TOTALS of criticals</strong></td>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
<td>48</td>
</tr>
</tbody>
</table>

* indicated critical characteristics

Configuration types are listed by number in Appendix B

Comparison ratings are relative to all classes
### 2.9.6 Class 6

<table>
<thead>
<tr>
<th>Attribute</th>
<th>127</th>
<th>128</th>
<th>129</th>
<th>130</th>
<th>131</th>
<th>132</th>
<th>133</th>
<th>134</th>
<th>135</th>
<th>136</th>
<th>137</th>
<th>138</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Availability</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Design Flexibility</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Fault Tolerance Comm.</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>6</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Recoverability</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Support Query Optimism.</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Support Dynamic Restruct</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Data locality Optimal</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Support Semantic Query</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>TOTALS of critical only</td>
<td>27</td>
<td>37</td>
<td>38</td>
<td>29</td>
<td>39</td>
<td>40</td>
<td>35</td>
<td>49</td>
<td>50</td>
<td>37</td>
<td>50</td>
<td>51</td>
</tr>
</tbody>
</table>

*indicated critical characteristics

<table>
<thead>
<tr>
<th>Attribute</th>
<th>139</th>
<th>140</th>
<th>141</th>
<th>142</th>
<th>143</th>
<th>144</th>
<th>145</th>
<th>146</th>
<th>147</th>
<th>148</th>
<th>149</th>
<th>150</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Availability</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Design Flexibility</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>3</td>
<td>9</td>
<td>9</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Fault Tolerance Comm.</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>Recoverability</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Support Dynamic Restruct</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Data locality Optimal</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Support Semantic Query</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>TOTALS of critical only</td>
<td>38</td>
<td>52</td>
<td>53</td>
<td>40</td>
<td>53</td>
<td>54</td>
<td>31</td>
<td>43</td>
<td>44</td>
<td>35</td>
<td>45</td>
<td>46</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>151</th>
<th>152</th>
<th>153</th>
<th>154</th>
<th>155</th>
<th>156</th>
<th>157</th>
<th>158</th>
<th>159</th>
<th>160</th>
<th>161</th>
<th>162</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Availability</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Processor Speedup</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Design Flexibility</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Fault Tolerance CPU</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Fault Tolerance Comm.</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Easily Support Log</td>
<td>8</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Recoverability</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Support Dynamic Restruct</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Support Monitoring</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Support Query Weighting</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Data locality Optimal</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>None Committed Data</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>9</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Support Semantic Query</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Total Time Speedup</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>TOTALS of critical only</td>
<td>12</td>
<td>55</td>
<td>57</td>
<td>43</td>
<td>56</td>
<td>57</td>
<td>44</td>
<td>58</td>
<td>59</td>
<td>44</td>
<td>57</td>
<td>58</td>
</tr>
</tbody>
</table>

*indicated critical characteristics

Configuration types are listed by number in Appendix B

Comparison ratings are relative to all classes
Chapter 3

Expert System Proposal and Analysis
3.1 Expert System Component Interfaces

Figure 3.1: Expert System Component Interfaces
3.2 Expert System Proposed Enhancements

- Support Dynamic Fragmentation.
- Reduce the Design/Redesign effort and overhead.
- Dynamic Query Optimization through Semantics.
- Expand flexibility of query language.
- Increase performance\(^1\) in both the distributed and non-distributed environments.
- Enhanced Node Recovery.

3.3 Expert System’s Key Processing Components.

The proposed Expert System will be presented in two fashions. First in a descriptive text format that verbally describes the components and the purpose for each. Secondly in a pseudo code format that does not require the syntactical knowledge of a specific language to understand more details of the process, and to be used for determining time complexity.

A point to be established before the reader continues is that in the commercial marketplace in any type of system several excellent monitoring products are available to analyze and make suggestions concerning performance and tuning. The key to this observation is that these components are an optional and expensive item. Although these products may produce information to improve systems performance much like an Expert System, they lack three major characteristics. First, since these products are optional and generally a third party vendor\(^2\) item they lack true connectivity which leads to inefficiencies. Second, instead of dynamically restructuring and tuning the system to increase performance, all actions must be manually translated and pass through logical evaluation again, even though the product may be very precise in its recommendations\(^3\). Third, but most important is that historical reporting if produced, is rarely used to tailor the system in attempts to restructure the hardware as

\(^1\)on some systems, depending on frequency of actions and dynamic nature of system
\(^2\)Even 'IBM business partners' lack this quality
\(^3\)Omegamon, Insight, IBM’s Bauchman Tools.
well as data allocations to attain optimal system performance. This last observation is made from experience not only in database related applications but also operating system and networking applications.

- **Transaction Manager Interface.**
  
The Transaction Manager Interface’s sole purpose is to accept queries to, and pass modified queries from the interface. It will accept by reference the query which is destined to be processed by the database backend processor, generally the QUEL or SQL processor. First stage packeting takes place and a unique transaction ID is assigned.

- **Initial Request Evaluator.**
  
The reason this module is not referred to as the *Initial Query Evaluator* is that there is the possibility that requests will not always be queries, it is necessary to support alter and report capabilities since the physical and logical structure will not always be what is perceived by the user. Second stage packeting occurs here to parse out original request and build a structure of the query conducive to coprocessing.

- **Semantic Query Optimizer.**
  
The Semantic Query Optimizer is in a sense the key component to the entire system. The Semantic Query Optimizer not only uses tuple relational algebra to guarantee optimum performance, but is also responsible for interfacing with the Prior Request Manager, the System Database Status Manager and Multiple Processor Status Manager to translate and packet the original query or action to reflect current physical and logical configuration.

  The Semantic Query Optimizer will parse the original request out into a usable form, translate the request into an equivalent action by using information from system tables and information stored in other tables by the Expert System to augment the functionality of these tables. Each action is logged and acted upon to ensure integrity in the actual logical structure. Once the action has been translated, appropriate actions taken, the management tables updated, and all actions logged, the translated action is inserted into the Internal Queue Manager’s queue with a priority to be passed to the QUEL or SQL processor⁴.

---

⁴Dependent upon implementation
• Internodal Request Manager.
Manages the transfer of requests between each of the nodes in the distributed version of the Expert System. It interfaces directly with the Internal Queue Manager at each node, and during remote node failure helps to coordinate the data recreation process set out by the Reconstruction Manager.

• Internal Time Manager.
Maintains activity keypoints for system, internal and internodal timeout condition checking. Interfaced by Prior Request Manager, and Internal Queue Manager to decide global system status.

• Internal Query Manager.
In a completely integrated Expert System\(^5\), the Internal Query Manager would handle request to SQL or QUEL backend on your behalf. It would also handle buffering of tasks in timesharing, this would allow several task to be concurrent, even in a single processor environment.

• Internal Queue Manager.
Handles the multi-task environment by prioritizing the request packets that have entered its queue. Also handles determination of packeting completion for root-children packets.

• System Database Manager.
This module and its subordinates maintain information about all tables and databases in the system. Such information as database and table location in the distributed system, last accessed information, lock information and like information are all maintained and updated by this module.

• Prior Request Manager.
This module and its subordinates are used to maintain relative action information, and pertinent information about prior requests and transient tables. This information is used to alter actions to reduce the redundant or partially redundant actions by joining transently held data by data retrieved to support the entire action. This module uses predetermined limits to determine length of residency and size of transient tables.

\(^5\)One that does not require the passing method as we modeled
• Log Manager.

Although it may seem that logging in a system does not justify a specific focus on log management, this module and its subordinates play a major role in maintaining transient integrity and remote site reconstruction at site failure. The Log Manager receives and returns logging requests by the processing system. It must work as an autonomous subsystem to all system modules.

• Reconstruction Manager.

The Reconstruction Manager sees its major impact and contribution when a remote node in the distributed system becomes unavailable in reference to the entire system. The Reconstruction Manager will be used to recreate a node's database by prior replication of data and using the Database Status Manager and log to create duplicate tables and databases to become available with full integrity until the lost node in the distributed system becomes available again.
3.4 Pseudo Code of Expert System

The following section represents the functions of the expert system in a more detailed fashion by describing the logical flow represented by a C type pseudo code. Figure 3.1 can be used to visualize the interactions of the modules representing the subfunctions of the expert system.

3.4.1 Transaction Manager

Transaction-Manager-Interface(pointer)
transaction-record-in *pointer;
/* Main duty of Transaction Manager Interface is to buffer the requests entering the Expert System at local node, from local transaction interface. It also handles the buffering of returned information (Pseudo code not shown here), so there are two distinct paths through the Manager which reduce the complexity.
Also creates first level of packeting and assigned unique transaction ID. */
{
/* global variable */
GLOBAL request-queue-pointer *rq-pointer;
if (pointer != NULL) /* must test for null request */
    Log-Request('initiaP,pointer);
    rq-pointer = Get-Space(length(pointer));
    /* get pointer to free space to store new request (possibly available buffer pool space or even C chained storage) */
    rq-pointer.transaction-request = pointer;
    rq-pointer.request-type = Simple-parse(pointer);
    rq-pointer.tranid = Unique-id();
    Initial-Request-Evaluator(rq-pointer);
else
    Error-routine('null-input',message-number,pointer);
    /* null input causing error */
} ;
Simple-parse(pointer)
char *pointer;
/* Simple parse returns query type, for direct parsing.
Since first word must always be a reserved word,
and will reflect operation request, (and also
misspellings ), would also allow acceptance of
RET for RETRIEVE,ect. */
{
    char hold-char;
    char *hold-pointer;
    char *first-word;
    int i;
    while (*hold-pointer != ") *holdpointer++; /* remove blanks */
        if (hold-char >= 'A' and hold-char <= 'Z') hold-char = hold-char — 'A' + 'a';
        /* convert to lower case for compare */
            switch (hold-char) {
                case 'r': if (( i = strcmp(hold-pointer,key(1).keyword)) > 0 )
                    return('R'); /* retrieve */
                        if (( i = strcmp(hold-pointer,key(2).keyword)) > 0 )
                            return('U'); /* replace */
                                return('E'); /* error... */
                break;
                case 'a': if (( i = strcmp(hold-pointer,key(3).keyword)) > 0 )
                    return('A'); /* append */
                        if (( i = strcmp(hold-pointer,key(10).keyword)) > 0 )
                            return('S'); /* system request 'ALTER' */
                                return('E'); /* error... */
                break;
                case 'd': if (( i = strcmp(hold-pointer,key(4).keyword)) > 0 )
                    return('D'); /* delete */
                        if (( i = strcmp(hold-pointer,key(8).keyword)) > 0 )
                            return('G'); /* define */
                                if (( i = strcmp(hold-pointer,key(9).keyword)) > 0 )
                                    return('V'); /* destroy */
                break;
            }
return('E'); /* error ... */
break;
case 'p':  if ((i = strcmp(hold-pointer,key(5).keyword)) > 0 )
    return('P'); /* print */
    return('E'); /* error ... */
break;
case 'c':  if ((i = strcmp(hold-pointer,key(6).keyword)) > 0 )
    return('C'); /* create */
    return('E'); /* error ... */
break;
case 'm':  if ((i = strcmp(hold-pointer,key(7).keyword)) > 0 )
    return('M'); /* modify */
    return('E'); /* error ... */
break;
case 'l':  if ((i = strcmp(hold-pointer,key(11).keyword)) > 0 )
    return('L'); /* reporting functions */
    return('E'); /* error ... */
break;
default:  return('E'); /* if none of above error also */
    break;
};

3.4.2 Initial Request Manager

Initial-Request-Evaluator(rq-pointer)
transaction-record-in *rq-pointer;
{
    int priority;
direct-type *new-request;
    switch (rq-pointer.request-type) {
        case 'R': new-request = Getspace-Retrieve(rq-pointer);
            break;
        case 'U': new-request = Getspace-Replace(rq-pointer);
            break;
    };
}
case 'A' : new-request = Getspace-Append(rq-pointer);
    break;
case 'D' : new-request = Getspace-Delete(rq-pointer);
    break;
case 'P' : new-request = Getspace-Print(rq-pointer);
    break;
case 'C' : new-request = Getspace-Create(rq-pointer);
    break;
case 'M' : new-request = Getspace-Modify(rq-pointer);
    break;
case 'G' : new-request = Getspace-Define(rq-pointer);
    break;
case 'V' : new-request = Getspace-Destroy(rq-pointer);
    break;
case 'S' : new-request = Getspace-System(rq-pointer);
    break;
    }

Log-Request('Evaluate',tran-id,rq-pointer);
    new-request = (Parse(rq-pointer)); /* parses request into packets labeled
    as database, table within database,
    predicates used in query, columns
    used in predicates, columns needed
    to service query ... Or creates
    system query packet */
    if (new-request.request-type == 'S' or new-request.request-type == 'L' )
        Prioritize-System-Request(new-request);
        Internal-Query-Manager('I',new-request);
    else
        Semantic-Query-Optimizer(new-request);
    } ;

46
Unique-id();
{
    /* each node will have a specific node number to be masked into transaction ID for originating node identification */
    int trannum;
    global int high-tran-id;
    high-tran-id++;
    trannum = Shiftright(hightran-id,2);
    return(trannum = Mask(01,Shiftright(trannum,2)));  
}

Prioritize-System-Request(new-request)
parse-system *new-request;
/* There are several types of 'system' related request necessary in a DB system. They must be separated out into groups so that certain 'groups' of request obtain highest overall system priority, even to extent of interrupting active requests. An example of high priority system request would be a request to stop all remote accesses, or terminate the subsystem, or manually purge out a transaction. Second level priority would be a display of system configuration for reference, manually increasing the priority of a specific request, or altering configuration defaults. */
{
    switch (new.request.system-request.action) {
        case '1' : return(450);
                    break;
        case '2' : return(475);
                    break;
        case '3' : return(500);
                    break;
        case '4' : return(525);
                    break;
        case '5' : return(550);
    }
}
break;
case '6': return(575);
break;
case '7': return(600);
break;
};

3.4.3 Log Manager

Log-Request(pointer)
log-record *pointer;
{
char *read-record;
char *return-code;
WRITE:
return-code = write-record(pointer);
if return-code is 'log is full'

.25in {
    return-code = SWITCH-LOG();
    if return-code is 'ok lvl 1'
        HALT-SYSTEM(return-code);
        /* logs full, I/O interface down */
    else {
        if return-code is 'ok lvl 2'
            return-code = SWITCH-LOG();
        else {
            if return-code is 'not ok'
                HALT-SYSTEM(return-code);
                /* problem switching logs */
            else
                GOTO WRITE;
        }
    }
else {


48
if return-code is 'good'
{
    return-code = READ-LOG-RECORD(record,'previous');
    if return-code = 'good'
    {
        return-code = COMPARE-LOG-RECORD(pointer,record);
        if return-code is 'not good'
        {
            return-code = SWITCH-LOG();
            if return-code is 'not good'
                HALT-SYSTEM(return-code);
            else
                LOG-REQUEST(101,log-write-error);
                GOTO WRITE;
        }
    }
    else /* compare is good */
    {
        return(return-code);
    }
}
else /* read of written log record is bad */
{
    return-code = SWITCH-LOG();
    if return-code is 'not good'
        HALT-SYSTEM(return-code);
    else {
        LOG-REQUEST(101,log-read-after-written-error,
        ignore possible previous log message);
        GOTO WRITE;
    }
}
else
    return(return-code);
Parse(pointer);
transaction-record-in *pointer;
{
    /* Requests will be parsed out into 'packets' used to:
       1) Optimize processing of request through Expert System
           (An attempt to reduce the overhead)
       2) Allow for multi-DB queries to be serviced in a system
           that may not currently support more than one DB to be
           used per query (INGRES, POSTGRES)
       3) Support the first level of translation for the dynamic
           structuring facility of the Expert System
       4) Prepare for manual semantic optimization if necessary,
           (The fact is that even some of the more advanced RDBMS
           do not attempt to restructure the initial query for
           efficiency )

       A parser could be built using LEX or hand written to
       parse a string based upon the semantic rules of a given
       language. We hand wrote a parser based upon the semantic
       rules given in section 4.1.5 for QUEL. (We wrote our parser
       by hand because of the overhead induced by such products as
       LEX). */

    The parser even though hand written and based upon section 4.1.5 parser
    requirements is really too long in even pseudo code to include here.
    If you understand the principles of compiler construction and could
    follow the Simple-Parsers workings, it is not hard to visualize
    what the full parser looks like. Also, the parser is highly
    subsystem dependent and based on the access method language, keywords,
    which change from system to system. The important point to realize is
    that an efficient parser could be generated for every system
} ;
3.4.4 Semantic Query Optimizer

Semantic-Query-Optimizer(new-request)
direct-type *new-request;
/* set priority of request based upon:
1) if verification of db, comm link ect.
2) process verification (is process still going?)
3) request for query only
   a) if attributes for query keys or not
      1) quantity of search limits
   b) amount of space needed to service
   c) if internodal request
   d) if part of a prior request that a temp dataset
      still exists
4) request for insert (must notify all nodes that
   have requested range that would include this insert)
5) request for replacement
   (must also notify the same as insert)
6) request for delete
   (must also notify the same as insert)

Implements techniques of semantic query optimization
through tuple relational calculus. Also works with
dependency rules to define minimum subquery structure,
taking into consideration the geographical locality
of the data.
Sends queries to Queue Manager, Designed as a root query,
and child queries where a "child query" is completed the
parent query is notified, when all dependent queries are
satisfied, then the root query is finished and a response is
produced. If a parent query finds that a child query is
taking too long and is holding up completion of entire
query then a request to bump up the priority on a dependent
"child" query is made.
*/
char *request-to-system, *response;
char answer;
transaction-root-identifier-link *converted-request;
/* shown only is the retrieve path, all others are similar but much simpler */
answer = Check-tables(new-request);
if ( answer = 'N')
    No-translate(new-request);

if (new-request.request-type = 'R')
{
    Temp-tbl = Chase-chains-retrieve(new-request.retrieve-request);
    converted-request = Build-response-retrieve(new-request.retrieve-request,
                            Temp-tbl);
    answer = System-db-status-manager('L',converted-request);
    if answer = 'N';
        Error('tables unavailable',108,converted-request);
        Reconstruction-Manager(answer);
        /* must be expanded to retry with other table combinations */
    if ( converted-request == NULL)
        Error('No data qualified found',109,converted-request);
    Assign-priority(converted-request);
    Prior-request-manager(converted-request);
    Internal-queue-manager('I',converted-request);
} ;

Chase-chains-retrieve(request)
parse-retrieve *request;
{/* first chase the predicate chain and build first stage list of tables that
are eligible for retrieval, then chase attribute list to create final list of
 tables needed to complete request service. Then determine smallest table to

search, then convert table names into packet equivalent table-names and quit. */
char *smallest;
char *Temp-T1, Temp-T2;
predicate-list *pred;
attribute-list *attrib;

pred = request.predicate-list;
if ( pred != NULL )
{
    Temp-T1 = Get-temp-table();
    Retrieve into unique Temp-T1( Eattribute.relid, Eattribute.attribute)
    where (Eattribute.table == pred.table or
    (Eattribute.relid == Erelation.tbl-name and Erelation.frg-of == pred.table))
    ( Eattribute.attribute == pred.attribute) and
    ( 'CONSTRAINT LIST' )
pred = pred.nxt-ptr;
    Do while ( pred != NULL )
    {
        Temp-T2 = Get-temp-table();
        Retrieve into unique Temp-T2 ( Eattribute.relid, Eattribute.attribute)
        where (Eattribute.table == pred.table or
        ( Eattribute.relid == Erelation.tbl-name and
        Erelation.frg-of == pred.table ) and
        ( Eattribute.relid == Temp-T1.relid ) and
        ( Eaatribute.attribute == pred.attribute ) and
        ( 'CONSTRAINT LIST' )
        Copy(Temp-T2, Temp-t1);
        pred = pred.nxt-ptr;
    }
};

/* Temp-T1 now contains reduced list of tables to search, now we need to
chase the attribute list to finalize all tables needed */

Do while ( attrib != NULL )
{

Append into unique Temp-T1 (Eattribute.relid, Eattribute.attribute)
where (Eattribute.table == attrib.table or
(Eattribute.relid == Erelation.tbl-name and
Erelation.frag-of == attrib.table)) and
(Eattribute.attribute == attrib.attribute)
attrib = attrib.nst-ptr;
}
return(Temp-T1);
} /* end of chase-chain-retrieve */

Build-response-retrieve(request,Temp-Tbl)
parse-request *request;
char * Temp-tbl;
{
int size;
size = Sizeof(Temp-tbl) /* size of deals with frags, if 2 tables in Temp-T1
but equivalent to 1 table, then counts as 1 table, the other routines must account for this */
if (size == 0) return(Null);
else
    return(Build-parent-child-retrieve(request,Temp-T1));
};

Build-parent-child-retrieve(request,Temp-tbl)
char *Temp-tbl;
parseretrieve *request;
{
extern-table * external-table;
char *table;
transaction-root-identifier-link *converted;
external-table = External-build(Temp-tbl); /* builds attributed related
for each packet */
return(converted = Translate-retrieve(external-table,request));
} ;
3.4.5 Priority Request Manager

Prior-request-manager(converted)
transaction-root-identifier-link *converted;
{
    /* i/o can also contain an acknowledgment this database
    will be used again.... date,time,other info

    Prior request manager not only tracks and maintains
    temporary database/fragment information, but decides
    if temporary space allocation for dataset is to be
    kept. Must work off of a dynamic 'system request'
    to alter amount of storage used for temporary
datasets. */

    int size;
    size = Calculate-size(converted);
    if (Eligible-for-temp(converted));
        Convert-to-into-select(converted) /* for creating transient tables */
        Replace (Etemp.size == Etemp.size - size ) where Etemp.table = 'Avail';
    }

Convert-to-into-select(converted)
transaction-root-identifier-link *converted;
{ /* convert plain retrieve into a ...
    RETRIEVE into TEMP-TABLE, and then
    SELECT TEMP-TABLE, so that transient table created
    actually these two operations could
    be going on at the same time if
    embedded into the RDBMS */
    } ;
3.4.6 System DB Status Manager

System-db-Status-Manager(Req,List)
char Req;
direct-request *List;
{
    /* tracks database/table fragmentation and geographical
distribution and uses the database record structure
to decide locality of data. Manages recovery or
reorganization process by maintaining attribute
information.
Uses the log manager, and works with reconstruction
manager on node failure to reconstruct the
database/table at another site.
*/

    if (Req == 'L') return (Lock-table(List));
    of (Req == 'U') return (Unlock-table(list));
    if (Req == 'F') return (Lock-table(List));
}
3.4.7 Internal Queue Manager

Internal-queue-manager(Req,pointer)
char Req;
pass-record *pointer;
/* must handle interface with internodal request manager,
'R' type records are from or to remote sites,
'I' type records are inserts into queue from local node
'A' are from the internal query manager for an answer
{
if ( Req == 'A' ) /* answer local or remote */
{
if ( tranid(pointer) == 'child' )
{
    subtract-one-from-parent(pointer);
    mark-tran-id-complete(pointer.tran-id);
}
if ( tranid(pointer) == 'parent' )
    Complete(tranid) /* send results to either internodal request manager
or transaction interface manager */
}
if ( Req == 'I' ) /* insert type */
{
    Insert-in-queue(pointer);
    Internal-query-manager(pointer.converted-request,tranid);
} ;
3.4.8 Internal Query Manager

Internal-query-manager(pointer,id)
char *pointer;
int id;
/* sends and receives request to Relation Database System on Expert Systems behalf.
Uses access control block structure of operating system to request services */
control-block-struct *answer;
answer RDBMS(pointer);
Internal-queue-manager('A',Reformatted(answer));
}

3.4.9 Reconstruction Manager

Reconstruction-manager(answer)
char *answer;
{
/*
initials and lends recovery of databases/tables
fragments after nodal failure or 'system request'
take-over for a predefined nodal outage or when
it is decided that a nodal failure has occurred.
*/
answer = Notify('table unavailable', Table-list);
if ( answer == 'wait' )
Wait('Time');
Reconstruction-manager(answer);
if ( answer == 'reconstruct' )
Recall-archives-alternate-site(Table-list);
    Rebuild(Table-list);
}
3.4.10 Internodal Request Manager

Internodal-request-manager(Req,pointer)
char Req;
char *pointer;

/* buffers and retrieves request to and from the other
geographically located nodes. Maintains necessary message
passing to secure a stable communications network. */

{
if ( Req == 'S' ) /* send */
Send-to(pointer.node,pointer);
if ( Req == 'R' ) /* receive */
Internal-queue-manager('A',pointer);
} ;
3.5 Record Types for Expert System

```c
typedef struct {
    char *transaction-request;
    char request-type;
    int tran-id;
} trans-record-in;

type transaction-record-in
{
    (char *transaction-request;
    char request-type;
    int tran-id;
    ) trans-record-in;

    type transaction-record-out
{
        (char request-type;
        char *transaction-answer;
        int tran-id;
        int return-code;
        ) trans-record-out;

    type direct-type
{
        (char request-type;
        parse-create *create-request;
        parse-retrieve *retrieve-request;
        parse-replace *replace-request;
        parse-append *append-request;
        parse-modify *modify-request;
        parse-delete *delete-request;
        parse-print *print-request;
        parse-define *define-request;
        parse-destroy *destroy-request;
        parse-alter *alter-request;
        parse-report *report-request;
        ) direct-record;

    type transaction-root-identifier-link
{
        transaction-root-identifier *root;
    transaction-child-identifier-link *first-child;
    ) trans-root-link-record;

```
type transaction-child-identifier-link {
    transaction-child-identifier *child;
    transaction-child-identifier-link *nxt-child;
} trans-root-link-record;

type parse-create { char request-type; int tran-id; char *node; char *database; char *table; char *prefix; frag-list *frag-link; temp-list *temp-link; int journaling; char *original-request; } parse-create-record;

type frag-list { frag-list *nxt-ptr; char *attribute; int structure; char *Lchar; char *Hchar; int Lint; int Hint; int Lcycle; int Hcycle; int key; } frag-record;

type temp-list { temp-list *nxt-ptr; char *attribute; int limit; } temp-record;
type parse-retrieve

(char request-type;
int tran-id;
int unique;
char *into-table;
attribute-list *attribute-link;
predicate-list *predicate-link;
) parse-retrieve-record;

type attribute-list

(attribute-list *nxt-ptr;
char *node;
char *database;
char *table;
char *prefix;
char *attribute;
) attribute-record;

type predicate-list

(predicate-list *nxt-ptr;
char *node;
char *database;
char *table;
char *prefix;
char *attribute;
constraint-list *constraint-ptr;
) predicate-record;

type preddel-list

(preddel-list *nxt-ptr;
char *attribute;
constraint-list *constraint-ptr;
) preddel-record;
type constraint-list
    (constraint-list *nxt-ptr;
     char operator;
     int Cint;
     char *Cchar;
     char *constraint;
    ) temp-record;

type parse-replace
    (char request-type;
     int tran-id;
     char *node;
     char *database;
     char *table;
     char *prefix;
     attrib-list *attribute-link;
     predicate-list *predicate-link;
    ) parse-replace-record;

type attrib-list
    (attrib-list *nxt-ptr;
     char *attribute;
     char *Cvalue;
     int Ivalue;
    ) attribute-record;

type parse-delete
    (char request-type;
     int tran-id;
     char *node;
     char *database;
     char *table;
     char *prefix;
     predicate-list *predicate-link;
    ) parse-delete-record;
type parse-append

( char request-type;
 int tran-id;
 char *node;
 char *database;
 char *table;
 char *prefix;
 attriba-list *attribute-link;
 predicate-list *predicate-link;
 ) parse-append-record;

type attriba-list

( attriba-list *nxt-ptr;
 char *attribute;
 char structure;
 char *default-spec;
 ) attriba-record;

type transaction-root-identifier

( char request-type;
 int tran-id;
 int priority;
 char *transaction-request; (original request)
 char *converted-request;
 (converted original request if does not break into child packets)
 int outstand-child;
 (outstanding child packets)
 )
type transaction-child-identifier
   (char request-type;
    int child-tran-id;
    int priority; ( must be larger than parent )
    char *transaction-request;
    (SQL statements to support packet)
    int outstand-child;
    (outstanding child packets)
   );

type extern-table
   (extern-table *nxt-table;
    int U-id;
    attlist *list;
   );

type attlist
   (attlist *nxt-list;
    char *attribute;
    char *table;
   );

type pass-record
   (char *node;
    char *service-type;
    char *request-type;
    int return-code;
    control-block *return-pointer;
    int tran-id;
   );
type db-status-record
{
  int locked-non-locked;
  char *node;
  char *database;
  char *table-name;
  char *attribute;
  char attribute-structure;
  int Linteger;
  int Hinteger;
 flt Lfloat;
  flt Hfloat;
  char *Lchar;
  char *Hchar;
};
3.6 Expert System Table Structures

Figure 3.2: Fragmentation Graphically Depicted
Enhancement : System Tables

<table>
<thead>
<tr>
<th>Table-Name</th>
<th>Level</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERELATION</td>
<td>1</td>
<td>Augment information stored about tables to support fragmentation and transient table management.</td>
</tr>
<tr>
<td>EATTRIB</td>
<td>2</td>
<td>Second level of fragment/transient table support, stores threshold information for attributes and HI/LOW's.</td>
</tr>
<tr>
<td>ECHAR-DATE</td>
<td>3</td>
<td>Splitting up the attribute stores for HI/LOW's : Char/Date formats.</td>
</tr>
<tr>
<td>EVARCHAR</td>
<td>3</td>
<td>Splitting up the attribute stores for HI/LOW's : Var-char formats only. (this is done to save space.)</td>
</tr>
<tr>
<td>DSTAT</td>
<td>0</td>
<td>Maintains global database and table information: (in some RDBMS this is already available)</td>
</tr>
<tr>
<td>ETTEMP</td>
<td>0</td>
<td>Maintain information on Transient Table range limits</td>
</tr>
</tbody>
</table>

### ERELATION

<table>
<thead>
<tr>
<th>Table-Name</th>
<th>Loc</th>
<th>Frag- #</th>
<th>Lowlimit</th>
<th>Highlimit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table-A</td>
<td>1</td>
<td>Table-A</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>Table-B</td>
<td>1</td>
<td>Table-A</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>Table-C</td>
<td>1</td>
<td>Table-A</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>Table-D</td>
<td>1</td>
<td>Table-C</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>Table-E</td>
<td>1</td>
<td>Table-C</td>
<td>40</td>
<td>50</td>
</tr>
<tr>
<td>Table-T1</td>
<td>1</td>
<td>Table-A</td>
<td>40</td>
<td>50</td>
</tr>
</tbody>
</table>

### EATTRIBUTE

<table>
<thead>
<tr>
<th>Table</th>
<th>Loc</th>
<th>U-id</th>
<th>relid</th>
<th>Attib</th>
<th>Type</th>
<th>Lihresh</th>
<th>Lycyle</th>
<th>Rihresh</th>
<th>Hcyyle</th>
<th>How</th>
<th>Shi</th>
<th>Flow</th>
<th>Fhi</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>01</td>
<td>Table-B</td>
<td>socsec</td>
<td>int</td>
<td>50,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>11212545626</td>
<td>298466250</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>02</td>
<td>Table-B</td>
<td>first-name</td>
<td>char</td>
<td>8,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>34010</td>
<td>721127</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>03</td>
<td>Table-B</td>
<td>last-name</td>
<td>vchar</td>
<td>8,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>0</td>
<td>31</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>04</td>
<td>Table-B</td>
<td>birth</td>
<td>int</td>
<td>1,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>34010</td>
<td>721127</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>05</td>
<td>Table-B</td>
<td>spouse-name</td>
<td>char</td>
<td>1,500</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>0</td>
<td>31</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>06</td>
<td>Table-B</td>
<td>eecode</td>
<td>int</td>
<td>2,000</td>
<td>1:00:35</td>
<td>4,000</td>
<td>0:01:00</td>
<td>1</td>
<td>4</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>07</td>
<td>Table-B</td>
<td>pycode</td>
<td>int</td>
<td>500</td>
<td>0:00:35</td>
<td>3,000</td>
<td>0:00:35</td>
<td>0</td>
<td>21</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>08</td>
<td>Table-D</td>
<td>socsec</td>
<td>int</td>
<td>50,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>2976752892</td>
<td>97569721</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>09</td>
<td>Table-D</td>
<td>first-name</td>
<td>char</td>
<td>8,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>0</td>
<td>31</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>10</td>
<td>Table-D</td>
<td>last-name</td>
<td>vchar</td>
<td>8,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>0</td>
<td>31</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>11</td>
<td>Table-D</td>
<td>birth</td>
<td>int</td>
<td>1,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>2976752892</td>
<td>97569721</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>12</td>
<td>Table-D</td>
<td>spouse-name</td>
<td>char</td>
<td>1,500</td>
<td>1:00:35</td>
<td>4,000</td>
<td>0:01:00</td>
<td>1</td>
<td>4</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>13</td>
<td>Table-E</td>
<td>eecode</td>
<td>int</td>
<td>1,500</td>
<td>6:00:35</td>
<td>4,000</td>
<td>0:01:00</td>
<td>0</td>
<td>31</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
<td>Table-E</td>
<td>pycode</td>
<td>int</td>
<td>500</td>
<td>0:00:35</td>
<td>3,000</td>
<td>0:00:35</td>
<td>0</td>
<td>21</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>15</td>
<td>Table-T1</td>
<td>socsec</td>
<td>int</td>
<td>50,000</td>
<td>2,000</td>
<td>0:01:00</td>
<td>0:000000000</td>
<td>999999999</td>
<td>b</td>
<td>b</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>16</td>
<td>Table-T1</td>
<td>first-name</td>
<td>char</td>
<td>8,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>0</td>
<td>31</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>17</td>
<td>Table-T1</td>
<td>last-name</td>
<td>vchar</td>
<td>8,000</td>
<td>1:00:35</td>
<td>2,000</td>
<td>0:01:00</td>
<td>0</td>
<td>31</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>
ECHAR-DATE

<table>
<thead>
<tr>
<th>U-id</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>Annie</td>
<td>Wilma</td>
</tr>
<tr>
<td>05</td>
<td>Allen</td>
<td>Zelda</td>
</tr>
<tr>
<td>09</td>
<td>Clara</td>
<td>Zelda</td>
</tr>
<tr>
<td>12</td>
<td>Andy</td>
<td>Joseph</td>
</tr>
<tr>
<td>16</td>
<td>Annie</td>
<td>Zelda</td>
</tr>
</tbody>
</table>

EVAR-CHAR

<table>
<thead>
<tr>
<th>U-id</th>
<th>Low</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>03</td>
<td>Augsburg</td>
<td>McCormack</td>
</tr>
<tr>
<td>10</td>
<td>Evans</td>
<td>Zellerbach</td>
</tr>
<tr>
<td>17</td>
<td>Augsburg</td>
<td>Zellerbach</td>
</tr>
</tbody>
</table>

DBSTAT

<table>
<thead>
<tr>
<th>Locked</th>
<th>Node</th>
<th>Database</th>
<th>Table</th>
<th>Attribute</th>
<th>Structure</th>
<th>Low-range</th>
<th>High-range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>A</td>
<td>DB1A</td>
<td>Table-A</td>
<td>secsec</td>
<td>int</td>
<td>110234512</td>
<td>925634243</td>
</tr>
<tr>
<td>Y</td>
<td>B</td>
<td>DB1B</td>
<td>Table-A</td>
<td>secsec</td>
<td>int</td>
<td>234566623</td>
<td>756541222</td>
</tr>
<tr>
<td>N</td>
<td>A</td>
<td>DB1A</td>
<td>Table-C</td>
<td>birth</td>
<td>int</td>
<td>340101</td>
<td>780304</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td>C</td>
<td>DB1C</td>
<td>Table-C</td>
<td>birth</td>
<td>int</td>
<td>350701</td>
<td>730213</td>
</tr>
</tbody>
</table>

ETEMP

<table>
<thead>
<tr>
<th>Table</th>
<th>Ret.</th>
<th>Attribute</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table-B</td>
<td>40</td>
<td>paycode</td>
<td>26,013</td>
</tr>
<tr>
<td>Table-B</td>
<td>40</td>
<td>paycode</td>
<td>48,024</td>
</tr>
</tbody>
</table>
3.7 Expert System Packeting

3.7.1 Packeting Structures

In the life of a request serviced through the Expert System interface the original request goes through several transformations so that the Expert System modules can manipulate the request. In designing the format, a structure that would allow for concurrent processing of the request was foremost in the design. The reader will notice that the design often times reflects several levels of chaining, this type of structure lends itself to a parallel or at least coprocessing environment.

There is a maximum of four forms of packeting under which the request can undergo. Each of the packet types and a transition to the next packet type are seen in the following figures.
**Structure of Packet chain for: Create**

<table>
<thead>
<tr>
<th>Type</th>
<th>Transid</th>
<th>Node</th>
<th>Database Name</th>
<th>Table Name</th>
<th>Fragment List</th>
<th>Temporary List</th>
<th>With/Without Journaling</th>
<th>Original Request</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Create Table-X** 
  \[ A1 = 12; \]
  \[ A2 = 21; \]
  \[ A3 = 14; \]
  \[ A4 = 129; \]
- **Participate** 
  \[ Temp,Frag] 
  \[ Frag (lowlimit = 40, highlimit = 50; Attribute = A1,thresh = 9,999, ...); \]
  \[ Attribute = A2,thresh = 250 ...); \]
  \[ Temp (constraint = A1, limit = 40, constraint = A2, limit = 20); \]

![Diagram of Packet Structure](image)

**Figure 3.3: Create Packet Structure**
Structure of packet chain for: Retrieve.

Figure 3.4: Retrieve Packet Structure
Figure 3.5: Replace Packet Structure
New attributes added to the table which are to also be considered in Fragmentation or Transient Tables must be added to the System Tables using the 'ALTER' command.

Figure 3.6: Append Packet Structure
Structure of Packet Chain for: Delete

Figure 3.7: Delete Packet Structure
Structure of the Internal Queue Packets:

Original request -
Replace (Table-A.eeo = 3)
where Table-A.eeo = 4;

(1) Packet created at Transaction Manager: (Assign request type and TransId)

(2) Internal Packet translation at Initial Request Evaluator: (Parse request)

(3) Translates to the following three packets for Internal Queue Manager
to service:

Figure 3.8: Transition Packet Stages
3.7.2 Packeting Transition Examples

In brevity the three main types of request types found in the Expert System interaction are shown. The remaining types of requests not show are similar in nature and packeting to the retrieve example.

The following three examples are show:

Example 1 Create of a Table.

Example 2 A retrieve request.

Example 3 A delete request.
3.7.3 Creating a Table

Creating a table within the database is quite similar to the regular processing except for the addition of the PARTICIPATE and FRAG and TEMP components to include the table in fragmentation and transient table usage. This inclusion or exclusion of the Participate key word will regulate which tables get considered for Expert System interaction, thus the System Administrator can decide when and how to tailor the system for optimal performance\(^6\).

**QUEL type request**

(1) Create CUSTOMER { firstname = C20;
    lastname = C20;
    paycode = I2;
    city = C20;
    payrate = I4; }

PARTICIPATE TEMP, FRAG
FRAG ( lowlimit = 40, highlimit = 50;
    attribute=firstname, hthreshold=9,999, hcycle=00:08:00, lthreshold=50, lcycle=05:00:00,
    key=N;
    attribute=lastname, hthreshold=9,999, hcycle=00:08:00, lthreshold=50, lcycle=05:00:00,
    key=N;
    attribute=city, hthreshold=200, hcycle=00:01:00, lthreshold=8,000, lcycle=00:08:00, key=Y;
    attribute=payrate, hthreshold=9,999, hcycle=00:08:00, lthreshold=2,000, lcycle=00:01:00,
    key=N;
    attribute=paycode, hthreshold=2,000, hcycle=00:01:00, lthreshold=9,999, lcycle=00:08:00,
    key=Y;)

TEMP (constraint = paycode, limit = 40;) /* \% of original table size */

/* if an attribute does not get explicitly specified in a FRAG or statement it still is put in the EATTRIBUTE system table but with no threshold or cycle values. This has to be done to relate the attributes to their physical placement when fragmented, depending on whether it uses the FRAG or TEMP statements. */

\(^6\) An alternate method to configure the Expert System would be to set 'default' thresholds and cycles for every attribute in the table. This would cause higher then necessary activity by the Expert System.
If it contains neither, the original request is maintained and entered only in the supportive RDBMS system tables, thus removing the need to maintain information about tables it does not manage. The following depicts the formats of the above QUEL statement.

Figure 3.9: Create Packet example Part 1
(3) Only the original request is put into the Internal Queue managers queue, the system table entries are append through direct QUEL/SQL statements without entering the internal queue.

```
Append {node.DB.}.ERELATION ( table = 'CUSTOMER';
  loc = node;
  frag-of = 'CUSTOMER';
)

Append {node.DB.}.EATTRIBUTE( table = 'CUSTOMER';
  loc = node;
  u-id = (generated unique id);
  relid = DB;
  attribute = 'firstname';
  type = Char;
  Ithreshold = 50;
  Icycle = 05:00:00;
  hthreshold = 9,999;
  hcyle = 00:08:00;
  key = N;
);

Append . . .

Append {node.DB.}.EVCHAR ( u-id = (generated unique id);
  low = NULL;
  high = NULL;
);

Append {node.DB.}.ETEMP ( table = 'CUSTOMER';
  attribute = 'paycode';
  rlimit = 40;
  size = 0;
);
```

Figure 3.10: Create Packet example Part 2
### 3.7.4 A Retrieve request

Using the table fields given in the system table example tables we show what could happen on a typical retrieve.

Original Request

1. Retrieve (Table-A.socsec) where Table-A.birth > 740101;

![Diagram of Retrieve Request]

(2) Attribute List

```
<table>
<thead>
<tr>
<th>R</th>
<th>17543</th>
<th>Null</th>
<th>Null</th>
<th>Attribute List</th>
</tr>
</thead>
</table>
| Ptr. node DB Table-A Prefix socsec
```

Predicate List

```
| Null | Null |
```

Retrieve (EATTRIBUTE.relid)

where EATTRIBUTE.table = Table-A and (EATTRIBUTE.attribute = birth and EATTRIBUTE.low < 740101 and EATTRIBUTE.high = 740101));

Retrieves only Table-D (no transient tables either)

(3) So one packet for Internal queue is generated:

```
<table>
<thead>
<tr>
<th>R</th>
<th>159</th>
<th>159</th>
</tr>
</thead>
<tbody>
<tr>
<td>R in Original Request</td>
<td>R in Converted Request</td>
<td></td>
</tr>
</tbody>
</table>
```

Retrieve (Table-D.socsec) where birth > 740101;

---

Figure 3.11: Retrieve Packet example
3.7.5 A Delete request

Again, we use the table fields given in the system table examples, we show the typical delete request.

Original Request

(1) Delete Table-A where Table-A.cco = 3;

Intermediate packet:

Retrieve (EATTRIBUTE.table) where (EATTRIBUTE.rclid = 'Table-A' and (EATTRIBUTE.attribute = eeo));

Returns tables Table-B and Table-E

(3) The following three packets get placed into the Internal Queue:

Delete Table-B where Table-B.cco = 3;

Delete Table-E where Table-E.cco = 3;

Figure 3.12: Delete Packet example
3.8 Examples of Primary Actions.

It is important in the understanding of our work that the reader be able to visualize the affects of the extended functionality offered by the Expert System. The presentation of six of the most widely implemented actions against this extended system are seen in the following examples. Notice that in the majority of the examples the reader will see reference to a single node. It should be stated that although the figure related to the example may only show a single site, it can be assumed that more then one site may have been involved in the actions of the example. This will be clearest when seen in examples three and six.

The following examples are used:

**Example 1** Database and Table creation.

**Example 2** Local Fragmentation after fragmentation threshold met.

**Example 3** Remote Fragmentation after fragmentation threshold met.

**Example 4** Maintaining Transient Tables under query only states.

**Example 5** Maintaining Transient Tables under update states.

**Example 6** Query servicing using a remotely fragmented database, ( distributed/fragmented table ).
3.8.1 Database and Table creation with Expert System

Using our EMPLOYEE database as an example we show how the interface would work from the beginning of creation of the database up through a selected period of time to show how the interface would determine optimal allocations. In the example we do not augment the initial load of the database with presumed information, as initially assigning indices and such but we do assign a primary key of employee-num to reduce unnecessary analysis by forcing the system to determine a primary key for you. Picking of the primary key is not necessary, as anomalies will not exist in the tuples of the table EMPLOYEE.

We assume that the interface thresholds have been set, database DB1 defined, table employee created and data loaded with primary key employee-num while this is proceeding, the system will collect the necessary information about the structure of the DB1 database and its associated table EMPLOYEE. A pictorial representation of the actions taken are seen in figures 3.2-3.6. Figure 3.13 shows the initial state of the system prior to creation of database DB1 and its sole table EMPLOYEE.

![Figure 3.13: Creating a Database](image)

Figure 3.13: Creating a Database
In figure 3.14 we see the process of loading the table EMPLOYEE, where the INGRES front end processes Applications By Forms (ABF) and the Operation Specification Language (OSL) are used to interface with the system enhancement through calls to a C procedure which handles the interface to the subcomponents of the system enhancement. The actual loading of the table EMPLOYEE is done by INGRES, so that he manages his own system datasets, but the interface is notified so that the system enhancement datasets for managing the databases and tables will be setup and managed.

![Diagram](image)

**Figure 3.14: Loading the Table**

Notice that although there may be several sites in the distributed system we load all data in a single table, at a single site. The question of why a determination of data locality was not considered is in order, but we use this example to show how the design portion of the the system handles an extreme case of poor planning.

The load and initialization of management datasets is important to guaranteeing optimal data allocation. Although this is time consuming, the benefits will be seen later. After the load of the table EMPLOYEE at Site A in the distributed system, Site A confirms the existence of table EMPLOYEE to all sites, and the database is made available to entire system.
As queries are serviced by a site, the interface determines by prior request information and available and transient DASD allocations best fit servicing. We take the following as an example of what might happen as several queries are serviced from a single site. Here we will have replication of data, managed by this interface and QUEL both, the use of transient data is determined by DASD availability and anticipation of further accesses. In a later example we will incorporate the determination of fragmentation, management of replicated data updates and other important information. Note that updates, inserts, ... to 'transient or replicated' table fragments need not be passed on to the alternate site, since only one current version (view) of the database is necessary to maintain integrity. Also notice that the interface uses predefined thresholds to determine attribute sensitivity.

Query One: Retrieve (EMPLOYEE.all)
where EMPLOYEE.union — status = 'Y'.
Results of the query are stored in Transient Table Q1.

Figure 3.15: Creation of Transient Table
Query Two: Retrieve (EMPLOYEE.first-name, EMPLOYEE.last-name, EMPLOYEE.payrate) where EMPLOYEE.department = 1394.
Results of the query are stored in Transient Table Q2 (shown in figure 3.16).

Figure 3.16: Creation of a second Transient Table
Query Three: Retrieve (EMPLOYEE.first-name, EMPLOYEE.last-name)
where EMPLOYEE.payrate > 500.00 and
EMPLOYEE.union — status = 'Y'
Results of the query are stored in Transient Table Q3 (shown in figure 3.17).

Figure 3.17: Creating Transient Table using previous Transient Table

At this point Q3 would use the transient table created by Q1. Payrate, dependent-care and union-status would all be noted as query constraints. Depending on the threshold of union-status, the tuples of EMPLOYEE could be fragmented to a new permanent table EMPLOY1 with with attributes union — status = 'N', and table EMPLOY2 with attributes union — status = 'Y', where it suffices to say that the maintenance of the transient data for Q1 improves performance of query reflective of tables initial size versus Q1's table size, where both extremes are important to note.
3.8.2 Local fragmentation after threshold met.

In this example we cover the process of fragmenting the table when the threshold is met. Figure 3.18 depicts the starting state where the table EMPLOYEE is loaded and defined to both INGRES and the system enhancement, and pictorially shows the presence of current transient tables. It is important to point out that in any relational database system a vast amount of DASD must be available for temporary tables and datasets used by the system and that the availability of additional DASD is not particularly necessary to obtain system functionality but it aids the performance dramatically. The larger the DASD allocation available for transient table use the less likely a future request will find that a transient table that could have been used to service the query or action has been removed to regain space for a more recent request. The system, by the way, is tailorble to handle such situations, with minimal design experience.

Figure 3.18: Initial Table Structure prior to Local Fragmentation
Figure 3.19 represents the occurrence of some query Q₁, where necessary information is stored in the system enhancements datasets and manages the creation of the transient table Q₁. During a nonspecific period of time, several other queries may also be passing through the system and being registered with the system enhancement.

Figure 3.19: Creation of Transient Table prior to Local Fragmentation
Figure 3.20 represents the state of the system after the creation of the two fragments EMPLOY1 and EMPLOY2, which are jointly equivalent to the contents of the original table EMPLOYEE. All actions to split EMPLOYEE into two fragments, arrange the primary and any secondary indices, register the database structure, and resolve any conflicts in constraints through views or the likes will be done by the system enhancement. Any further reference to EMPLOYEE will be converted to its equivalent reference to the respective fragments EMPLOY1 and EMPLOY2.

Figure 3.20: Threshold Met, Local Fragmentation occurs
3.8.3 Remote Fragmentation after fragmentation threshold met.

This example depicts queries from remote sites, the creation of a locally managed transient table, and the eventual creation of a remotely distributed fragment of the original table EMPLOYEE to increase performance through fragmentation for data locality. This would by far be one of the most common actions in the beginning of establishment of a system, and only later on as the system dynamically changes to suit the users processing needs. Figure 3.21 depicts the original state of the system prior to queries that will eventually lead to the fragmentation of the table EMPLOYEE.

Figure 3.21: Initial Table Structure prior to Remote Fragmentation
Figure 3.22 represents the state of the system after some period of time, reflecting the presence of at least one transient table $Q_1$, where $Q_1$ is equal to a fragment of EMPLOYEE where EMPLOYEE.city is equal to 'Henderson'. As other actions are taken against the system, the system enhancement logs these actions and makes a decision on whether EMPLOYEE has reached a fragmentation threshold.

Figure 3.22: Creation of Transient Table on predicate, prior to Remote Fragmentation
Figure 3.23 depicts the system state after fragmentation threshold has been reached. The attribute that forced the fragmentation was 'City', which would be an obvious attribute for driving data locality fragmentation. Notice from figure 3.23 that the alternate site for database DB1 / table EMPLOYEE, has been notified of the fragmentation and that all DB status managers at all sites will have been updated to reflect the new configuration.

Figure 3.23: Threshold Met, Remote Fragmentation occurs
3.8.4 Maintaining Transient Tables under query only states.

This example follows the states of the system as updates are done against a table where a transient table also exists with a major portion of the original table, or at least a large enough portion that a threshold has deemed that it would be feasible to also update this table to keep it in a synchronized state relative to the original table. The alternative when update actions are taken against the original table is to remove the transient table, since it will no longer be synchronized with the original. In figure 3.24 we show the state of the system with original table EMPLOYEE, and transient table Q1.

---

Figure 3.24: Initial Table Structures during Query-only state
Several actions may have taken place since the creation of Q1, but Q1 is still maintained, probably because there was sufficient transient table space. Query Q2 enters the system, where Q2 is some action where a constraint is that 'Dept = 1300'. Q1 can be used to reduce the search otherwise necessary when processing EMPLOYEE, in which case if EMPLOYEE is quite large, and if the constraint of Q1 was not a key this could dramatically improve Q2’s response time (seen here in Figure 3.25).

Figure 3.25: Additional Transient Tables created from prior request
3.8.5 Maintaining Transient Tables under update states.

Actions such as updates represent a small percentage of the total actions against any database system, but when they happen the system must be able to maintain and service actions when a table has been fragmented. Figure 3.26 depicts the state of the system prior to any action against the table.

Figure 3.26: Initial Table structures during Update state
Figure 3.27 depicts the state after query Q1 is serviced, the internal actions that must happen to obtain this state include converting the original query to a logically equivalent against EMPLOY1, EMPLOY2 or both, and to update the system datasets to reflect the transient table Q1.

Figure 3.27: Fragmentation in the Update state
3.8.6 Query Servicing using a remotely fragmented database.

This example, more than the previous, depicts a multi-node environment that any prior. Here we show what happens when a distributedly fragmented table is used in servicing a query which requires remotely stored data. In figure 3.28 we see the state of the system after its first level of fragmentation to create EMPLOYA and EMPLOYB showing the interactions of EMPLOYA, EMPLOYB, and the alternate site.

Figure 3.28: Query service in remotely fragmented database
As the end user requests a service that requires table access at the site containing only EMPLOYA, that site's system enhancement makes the determination when converting the original query that remote data is needed from a site containing EMPLOYB, thus sending its necessary subquery Q1 to that site for service. At the time of the request to EMPLOYB's site, the originating site will notify the alternate site of its intentions. At EMPLOYB's site query Q1 is handled just as if the request for action was local, except that the system datasets will reflect a remote request build and that the contents of the transient table must be passed to EMPLOYA's site. EMPLOYB's site will also, upon passing the contents of Q1 to the requesting site, notify the alternate site (seen in figure 3.29).

Figure 3.29: Transient Tables built to service remote request
Chapter 4

Definitions

4.1 RDBMS Structures and Implementations.

4.1.1 The Relational Database Defined.

To fully understand the impact of this work it is necessary to be familiar with the structures and implementations of the relational database. If you are familiar with such features of the relational database environment you may want to skip to the next subsection.

Because this paper is addressing a issue of performance and integrity and is not specifically a theoretical approach it was necessary to first understand what commercially available features were presently available, not only to avoid repetition of work but also to derive a knowledge of what attributes were already commercially in place to draw from when determining the feasibility of the work and to also help to develop an interface into a working system when the modeling took place.

There are several database systems commercially available in today's market, all of which can be safely grouped into three categories. The definition of a database is such that any data stored in a predetermined format would qualify as a database. This paper and its approaches to improving performance and integrity in the database environment are targeted at the relational form of database storage and retrieval, but as the reader may see, some of the byproducts of the main work may be implemented into nonrelational database systems.
4.1.2 The Relational Database Structure.

Here and throughout this paper we will use the INGRES and INGRES/star Relational Database Systems as references when such topics as structure, features and interfaces are discussed. The reason for choosing INGRES as a reference is because of its availability to the author, and INGRES/star because of its basic similarities to the nondistributed version.

A detailed understanding of what a relational database is is not necessary to understand the material presented in this paper, instead let it suffice to state that a relational database is nothing more then a database that is perceived by its users as a collection of tables (and nothing but tables).

There are a few attributes of the relational database structure that become significant by this definition.

1. All the data values are atomic. This means that there is exactly one data value for each row and column position in the table, never a set of values or a pointer to a linked list.

2. Information stored in the tables are defined in specific data types and currently the user is not allowed to create a user defined data type. Generally available data types include; character, integer, floating point, and two variations of integer and floating point known as money and date.

3. Generally a table contains one attribute (column) which acts to uniquely identify each row in the table. This is not necessarily a requirement for design, but nevertheless is recommended and generally accepted as a design rule. This unique identifier is referred to as the primary key, where there may be several key fields by which to refer to the data the primary key satisfies the requisite of uniqueness.
Figure 4.1 visually portrays the structure of a relation database. This format is used when servicing a query against a relational database in keeping with its predefined logical format. The physical format of the relational database is actually quite different. In the INGRES system, tuples are stored consisting of a stored record prefix, plus the stored attributes that make up the tuple. Each tuple is stored wholly on a single page, with a maximum total length of 2008 bytes. Although one tuple must be stored wholly on one page, there is no limit to the number of tuples per page.
4.1.3 The Relational Database Implementation.

The implementation of a database is determined by the flexability and usage available by using such a system. From the beginning when Codd defined the relational database specific data manipulation rules where derived, mainly from Codd's definition of the normal forms, the implied atomicity of the relational structure, and the underlying restrictions placed by these simplified data structures. Please see Appendix A for a full discussion of normal forms.

In the previous subsection we defined the logical and physical design of the relational database. In this subsection it is important that the format and methods for retrieving data will be discussed. As with any database structure we are not concerned with data manipulation or the calculation of database or nondatabase attributes. Although several commercially available relational systems offer frontend processes to accommodate this, it is not part of the formal definition of the relational system. The actions that are considered primary to the implementation of the relational system are the abilities to:

1. Create databases, tables and their attributes.

2. Add new tuples to a table.

3. Delete tuples from a table.

4. Replace tuples or attributes from a table.

5. Alter the structure of a table. This means that a user must be able to add or remove the occurrence of an attribute or attributes from all tuples of a table, thus altering the structure of the table.

6. Delete tables and databases.

A complete reference to action verbs relative to the relational database are given in subsection 4.1.5. It will be assumed that the reader has read this subsection or is familiar with action verbs used in relational queries in the following sections.
4.1.4 Restrictions on Relational Databases.

The previous subsections discussed a few data types and data manipulation restrictions inherent to a relational system, but there are many more inflexibilities inherent to most relational systems\(^1\). Although several of these restrictions will be removed by the implementation of our system enhancement, it was not our main goal for performance and data integrity improvement, rather a byproduct of its implementation.

\(^1\)References used here are those relative to the INGRES and INGRES-star systems, if other systems are exceptions to statements made it will be noted as such
• The following is a list of inflexibilities relative to the commercially available relational systems.

1. Requirement that each attribute be atomic\(^2\).
2. Limited amount of data types. This is a limit currently in most relational systems, one exception is POSTGRES which allows for some user defined types.
3. Single database updates at one time.
4. Limited fragmentation handling, this is currently limited to explicit fragmentation and replication.
5. Limited data retrieval optimization\(^3\). Most commercial systems use one of two techniques for data retrieval. One, sequentially search the entire database for the information needed, or use an index to retrieve the data if a key has been given as a search parameter.
6. Limited comprehensive information regarding attributes in a table. This would include such information as range of the attribute, level of activity of the attribute, and other key statistical information. Collecting this information through monitoring could be used to recommend implicit fragmentation or creation of indices.
7. Only explicit maintenance of relations used to service queries \(^4\). This is relative to tables built to service queries, which generally are released after completion unless otherwise stated.
8. The attributes used in the compare while creating a join must be the same, either both numeric, both strings, both money, or both dates. The data types do not necessarily need to be the same but should be for performance reasons.
10. Currently does not support referential key integrity\(^5\).

---

\(^2\)A attribute could uniquely identify a tuple in another table that would allow several occurrences, arrays etc. This would be managed by the Expert System

\(^3\)We are improving here by tracking requests and implicitly creating fragments and indices for data

\(^4\)Our enhancement relies heavily on the fact that most systems do not maintain this information and it is used to improve performance

\(^5\)IBM does as of release 2.1.0 of DB2
4.1.5 Parser Requirements.

Primary Keywords:

CREATE
DESTROY
DEFINE
DELETE
APPEND
MODIFY
PRINT
RETRIEVE
REPLACE

Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR(n)</td>
<td>char string</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>char string</td>
</tr>
<tr>
<td>I1,I2,I4</td>
<td>binary integer 1,2,4 bytes</td>
</tr>
<tr>
<td>F4,F8</td>
<td>floating point 4,8 bytes</td>
</tr>
<tr>
<td>MONEY</td>
<td>decimal currency two digits on right</td>
</tr>
<tr>
<td>DATE</td>
<td>char string max 12 bytes, (specific formats, see QUEL reference guide)</td>
</tr>
</tbody>
</table>

Aggregates

<table>
<thead>
<tr>
<th>Aggregate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNT</td>
<td>number in column</td>
</tr>
<tr>
<td>SUM</td>
<td>sum of values in column</td>
</tr>
<tr>
<td>AVG</td>
<td>average of values in column</td>
</tr>
<tr>
<td>MAX</td>
<td>largest value in column</td>
</tr>
<tr>
<td>MIN</td>
<td>smallest value in column</td>
</tr>
</tbody>
</table>
Relative Operators

=  equals
>  greater than
<  less than
>= greater than or equal to
<= less than or equal to
<> not equal to

CREATE format:

Create table-name ( column-definition { , column-definition } ... )
{ WITH JOURNALING }

with Expert System
Create table-name ( column-definition { , column-definition } ... )
{ WITH JOURNALING }
PARTICIPATE { TEMP,FRAG }
FRAG ( LOWLIMIT = value, HIGHLIMIT = value\(^6\) 
{ attribute-list, { attribute-list } ... } )
TEMP ( CONSTRAINT = attribute-name, LIMIT = value\(^7\) )

COLUMN-DEFINITION format:

data-type { default-spec }

DEFAULT-SPEC format:

WITH NULL || NOT NULL || NOT NULL WITH DEFAULT

\(^6\)in percentage
\(^7\)percentage of total table size before retain
ATTRIBUTE-LIST format:

\[ (\text{ATTRIBUTE} = \text{attribute-name}, \text{HTHRESHOLD} = \text{value}, \text{HCYCLE} = \text{value}, \text{LTHRESHOLD} = \text{value}, \text{LCYCLE} = \text{value}, \text{KEY} = \text{'Y/N'}) \]

DESTROY format:

\[ \text{DESTROY database} \parallel \text{DESTROY view} \parallel \text{DESTROY PERMIT table} \]

with Expert System

no change, deleting the original table or database will delete all related transient tables and fragmented sections.

RETRIEVE format:

\[
\text{RETRIEVE} \{ \text{INTO table} \} \{ \text{UNIQUE} \} \{ \text{target-list} \}
\{ \text{WHERE predicate} \}
\{ \text{SORTED BY attributes} \}
\]

with Expert System

there is no user change to this command yet

an internal translation may take place

TARGET-LIST format:

\[
\{ \text{unqualified-name} = \} \text{scalar-expression}
\]

SCALAR-EXPRESSION format:

\[
\text{scalar} \parallel \text{simple-aggregate}
\]

SCALAR format:

\[
\text{table.attribute} \parallel \text{constant}
\]
SIMPLE-AGGREGATE format:

\[ \text{aggregate} \left( \text{scalar-expression} \{ \text{WHERE} \ \text{predicate} \} \right) \]

PRINT format:

\[ \text{PRINT table} \]
\[ \text{with Expert System} \]
\[ \text{there is no user change to this command} \]
\[ \text{yet} \]
\[ \text{an internal translation may take place} \]

DEFINE format:

\[ \text{DEFINE INTEGRITY} \ \text{ON} \ \text{table IS} \ \text{table.attribute} \]
\[ \text{relative-operator scalar} \]
\[ \text{with Expert System} \]
\[ \text{there is no user change to this command} \]
\[ \text{yet} \]
\[ \text{an internal translation may take place} \]

\[ \text{DEFINE VIEW} \ \text{view} \ ( \ \text{target-list} ) \]
\[ \{ \ \text{WHERE} \ \text{predicate} \} \]
\[ \text{with Expert System} \]
\[ \text{there is no user change to this command} \]
\[ \text{yet} \]
\[ \text{an internal translation may take place} \]

\[ \text{DEFINE PERMIT} \ \text{operation(s)} \]
\[ \text{ON} \ \text{table} \ \{ \ ( \ \text{attribute} \ \{ , \ \text{attribute} \} \ ... \ ) \} \]
\[ \text{TO} \ \text{user} \]
\[ \{ \ \text{AT} \ \text{terminal(s)} \} \]
\[ \{ \ \text{FROM} \ \text{time1 TO} \ \text{time2} \} \]
\[ \{ \ \text{ON} \ \text{day1 TO} \ \text{day2} \} \]
\[ \{ \ \text{WHERE} \ \text{predicate} \} \]
with Expert System
there is no user change to this command
yet
an internal translation may take place

APPEND format:

APPEND TO table { UNIQUE } ( target-list )
{ WHERE predicate }

with Expert System
there is no user change to this command
yet
an internal translation may take place

MODIFY format:

MODIFY table TO spec { UNIQUE }
{ { ON attribute { direction } },attribute { direction } ... } 

with Expert System
there is no user change to this command
yet
an internal translation may take place

SPEC format:

{C}HEAP {C}HEAPSORT {C}ISAM {C}HASH {C}BTREE

REPLACE format:

REPLACE range-variable ( target-list )
{ WHERE predicate }

with Expert System
there is no user change to this command
yet
an internal translation may take place
DELETE format:

```
DELETE range-variable
{ WHERE predicate }
```

with Expert System
there is no user change to this command
yet
an internal translation may take place

INDEX format:

```
INDEX ON table IS index ( attribute { ,attribute } ... )
```

with Expert System
there is no user change to this command
yet
an internal translation may take place
index is a table name

ALTER format: (Expert System)

```
ALTER command-list (node,database-list,table-list)
ALTER RECOVER (node,database-list,table-list,time-stamp)
ALTER PARTICIPATE (node,database,table)
FRAGUPDATE (LOWLIMIT = value, HLIMIT = value (attribute-list))
ALTER PARTICIPATE (node,database,table)
TEMPUPDATE (constraint-list {constraint-list,})
```

COMMAND-LIST format:

```
LIST || LOCK || UNLOCK || STATUS
```
DATABASE-LIST format:

\[ \text{database-name} \parallel (\text{database-name}, \{\text{database-name}, ...\}) \parallel ^8 \]

TABLE-LIST format:

\[ \text{table-name} \parallel (\text{table-name}, \{\text{table-name}, ...\}) \parallel ^9 \]

CONSTRAINT-LIST format:

\[ \text{CONSTRAINT} = \text{attribute}, \text{LIMIT} = \text{value} \{, \text{CONSTRAINT} = \text{attribute}, \text{LIMIT} = \text{value}\}... \]
Chapter 5

Contributions and Conclusion

5.1 General Contributions of Paper

Although several commercial relational DBMS systems offer the user flexible manipulative functions including distribution of databases and replications for increased performance, we have concluded that simple interfaces could be easily installed into these commercially available versions that will enhance user functionality, data availability, and response time, all while being completely transparent to the end user. The system which we have chosen as a basis for modeling our enhancement interface is the INGRES DBMS, a more suitable system would have been the INGRES/STAR system as it is a distributed DBMS, but nevertheless it was not available for modeling so the enhancements were modeled on the non-distributed version. Several features which are actually not available from the non-distributed version will be assumed, and several features which are documented as future enhancements will also be assumed as available, even if not presently so.
The basics that add to the functionality of a distributed database from INGRES/STAR not available from INGRES are:

1. System transparency.
2. Location transparency.
3. Improved productivity.
4. Site autonomy.
5. Increased capacity and performance.

At the time of print of our references about INGRES/STAR, there were the following features to be available:

1. Additional CPU and operating systems support.
2. Additional network support.
3. Support of "SQL-based" companions\(^1\)
4. Multiple database updates\(^2\).
5. Data replication
6. Data fragmentation

\(^1\)IBM's DB2 and SQL/DS
\(^2\)Currently only one database can be updated at a time
We hope to offer a solution to several of the present and possibly future shortcomings of INGRES-INGRES/STAR and at the same time increase data availability and retrieval response time.

Our intentions are to offer a front-end processor that can be integrated into INGRES-INGRES/STAR and should be portable across any RDBMS with little adaptation necessary. A list follows that contains the contributions of this paper.

1. Enhanced retrieval time.
2. Enhanced data availability in presence of node failure.
3. Dynamic restructuring of database for optimal response and data integrity.
4. Support table fragmentation across sites.
5. Support multiple table updates in single query.
5.2 Conclusion

5.2.1 Chapter 2: Interdependencies

Comparing permutations of attributes of a system within a class will allow the system support staff the opportunity to reasonably compare alternative configurations to possibly improve the performance of that system. This type of comparison is a generalized approach and is not guaranteed to return optimum results for all configurations but can be used to reduce time spent by the support staff to benchmark several permutations to predict optimal performance. The matrices can be used to assist in narrowing the modeling configurations down to a handful of most likely candidate configurations and thus reduce cost of establishing a resulting configuration.

A complete understanding of interdependencies of the mentioned attributes of a distributed system can be gained by reading the referenced material. Papers of most value relate to topics of query optimization, fragmentation strategies and fault tolerance topics.

5.2.2 Chapter 3: Expert System

Low cost enhancement incorporated, or placed as a front-end to any Relational Database System can reduce the design/redesign process. Expected performance improvements and increased data integrity are also functional byproducts of the Expert System. Although the addition of the Expert System to the Relational System will increase overhead we feel that the performance benefit returned by the handling of the semantics, query frequencies and dynamic system restructuring by the Expert System will return a substantial amount of the overhead cost. In addition, the flexibility of this dynamic structure is so powerful that we feel until modeling is complete the true extent of this proposal can not be realized.

The consideration of transient tables, and the implicit fragmentation of queries, are expected to return improved response time dependent upon the query semantics and query frequencies used.
5.3 Future Work

In this paper we have proposed a method for improving performance through the use of transient tables, dynamic fragmentation schemes and inferred semantic optimization through a frontend enhancement. Embedding this process into the RDBMS could show increased performance through the removal of redundancy.
Appendix A

Normal Forms

There are several normal forms that have been defined in recent years, but the following 4 normal forms are the accepted basis of the industry for removing anomalies and redundancy in the relational database system [Ullman 82].

First Normal: Requires that the domain of each attribute consists of indivisible values, not sets or tuples of values from a more elementary domain or domains.

Second Normal: If a Relation $R$ has no partial dependencies$^1$, $R$ is in second normal form.

Third Normal: Relation $R$ is in third normal form if an attribute $X$ functionally determines attribute $A$ and $A$ is not in $X$, and $X$ is a superkey or $A$ is prime.

Fourth Normal: A generalization of Boyce-Codd that applies to multivalued dependencies. Relation $R$ is in fourth normal form if $Y$ has a multivalued functional dependency on $X$, and $Y$ is nonempty or a subset of $X$, and the union of $X$ and $Y$ are not in $R$, then $X$ is a superkey of $R$. 

\[\text{\footnotesize$^1$although it could have transitive dependencies}\]
Boyce-Codd: For relation \( R \), all attributes \( a_1, a_2, \ldots, a_n \) are functionally dependent on some attribute or attributes \( X \), such that \( a_1, a_2, \ldots, a_n \) are not in \( X \), the \( X \) is said to be a superkey for relation \( R \).

Superkey: \( X \) is a superkey if \( X \) is a key or contains a key.

Prime: Attribute \( A \) is prime if it is a member of any key, otherwise \( A \) is non-prime.

Partial

Dependency: If \( X \) is a proper subset of a key and \( A \) is functionally dependent on \( X \) then this relation is considered a partial dependency.

Transitive

Dependency: If \( X \) is not a proper subset of any key, and \( Y \) is a key, then \( Y \Rightarrow X \Rightarrow A \), which says that if \( X \) functionally determines \( A \) and \( Y \) functionally determines \( X \) then \( Y \) also functionally determines \( A \).
Appendix B

Hardware Configurations

B.1 Single node system with M sites, $M \geq 1$. (CLASS 1)

1. Single node system with M sites, where $M \geq 1$. System has multiple processors. Queries are serviced through server node. System supports communications through node to node communications. Processors are tightly-coupled.

2. Single node system with M sites, where $M \geq 1$. System has multiple processors. Queries are serviced through server node. System supports communications through node to node communications. Processors are loosely-coupled.

3. Single node system with M sites, where $M \geq 1$. System has single processor. Queries are serviced through server node. System supports communications through node to node communications. Processors are tightly-coupled.

4. Single node system with M sites, where $M \geq 1$. System has single processor. Queries are serviced through server node. System supports communications through node to node communications. Processors are loosely-coupled.

5. Single node system with M sites, where $M \geq 1$. System has multiple processors. Queries are serviced through server node. System supports communications by token ring through multiple paths. Processors are tightly-coupled.

6. Single node system with M sites, where $M \geq 1$. System has multiple processors. Queries are serviced through server node. System supports communications by token ring through multiple paths. Processors are loosely-coupled.
7. Single node system with $M$ sites, where $M \geq 1$. System has single processor. Queries are serviced through server node. System supports communications by token ring through multiple paths. Processors are tightly-coupled.

8. Single node system with $M$ sites, where $M \geq 1$. System has single processor. Queries are serviced through server node. System supports communications by token ring through multiple paths. Processors are loosely-coupled.

9. Single node system with $M$ sites, where $M \geq 1$. System has multiple processors. Queries are serviced through server node. System supports communications by token ring through minimized paths. Processors are tightly-coupled.

10. Single node system with $M$ sites, where $M \geq 1$. System has multiple processors. Queries are serviced through server node. System supports communications by token ring through minimized paths. Processors are loosely-coupled.

11. Single node system with $M$ sites, where $M \geq 1$. System has single processor. Queries are serviced through server node. System supports communications by token ring through minimized paths. Processors are tightly-coupled.

12. Single node system with $M$ sites, where $M \geq 1$. System has single processor. Queries are serviced through server node. System supports communications by token ring through minimized paths. Processors are loosely-coupled.

B.2 Single processor per node, multi-node. (CLASS 2)

13. Single processor per node, multi node system. Queries are serviced through server node. System supports communications through node to node communications.

14. Single processor per node, multi node system. Queries are serviced through server node. System supports communications by token ring through minimized paths.

15. Single processor per node, multi node system. Queries are serviced through server node. System supports communications by token ring through multiple paths.
16. Single processor per node, multi node system. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications.

17. Single processor per node, multi node system. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths.

18. Single processor per node, multi node system. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths.

B.3 Multi-processor, multi-node with \( P > N, P < N \times 2 \). (CLASS 3)

19. Multiple processors, Multiple nodes with \( P > N \) but \( P < N \times 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Non shared storage.

20. Multiple processors, Multiple nodes with \( P > N \) but \( P < N \times 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Shared storage.

21. Multiple processors, Multiple nodes with \( P > N \) but \( P < N \times 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Minimized shared storage.

22. Multiple processors, Multiple nodes with \( P > N \) but \( P < N \times 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Non shared storage.

23. Multiple processors, Multiple nodes with \( P > N \) but \( P < N \times 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Shared storage.

24. Multiple processors, Multiple nodes with \( P > N \) but \( P < N \times 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Minimized shared storage.
25. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring

26. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring
through minimized paths. Processors loosely coupled. Shared storage.

27. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring
through minimized paths. Processors loosely coupled. Minimized shared storage.

28. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring
through minimized paths. Processors tightly coupled. Non shared storage.

29. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring
through minimized paths. Processors tightly coupled. Shared storage.

30. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring
through minimized paths. Processors tightly coupled. Minimized shared storage.

31. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring

32. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring
through multiple paths. Processors loosely coupled. Shared storage.

33. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring
through multiple paths. Processors loosely coupled. Minimized shared storage.

34. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are
serviced through host node. System supports communications by token ring
35. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Shared storage.

36. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Minimized shared storage.

37. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Non shared storage.

38. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Shared storage.

39. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Minimized shared storage.

40. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Non shared storage.

41. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Shared storage.

42. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Minimized shared storage.

43. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System
supports communications by token ring through minimized paths. Processors loosely coupled. Non shared storage.

44. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors loosely coupled. Shared storage.

45. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors loosely coupled. Minimized shared storage.

46. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Non shared storage.

47. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Shared storage.

48. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Minimized shared storage.

49. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors loosely coupled. Non shared storage.

50. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors loosely coupled. Shared storage.

51. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System
supports communications by token ring through multiple paths. Processors loosely coupled. Minimized shared storage.

52. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Non shared storage.

53. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Shared storage.

54. Multiple processors, Multiple nodes with $P > N$ but $P < N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Minimized shared storage.

55. P processors, and N nodes where $P = N \times 2$.

\textit{(CLASS 4)}

56. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Shared storage.

57. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Minimized shared storage.

58. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Non shared storage.
59. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Shared storage.

60. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Minimized shared storage.

61. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors loosely coupled. Non shared storage.

62. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors loosely coupled. Shared storage.

63. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors loosely coupled. Minimized shared storage.

64. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors loosely coupled. Non shared storage.

65. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors tightly coupled. Shared storage.

66. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors tightly coupled. Minimized shared storage.

67. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors loosely coupled. Non shared storage.

68. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors loosely coupled. Shared storage.
69. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors loosely coupled. Minimized shared storage.

70. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Non shared storage.

71. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Shared storage.

72. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Minimized shared storage.

73. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Non shared storage.

74. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Shared storage.

75. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Minimized shared storage.

76. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Non shared storage.

77. P processors, and N nodes where \( P = N \times 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Shared storage.
78. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Minimized shared storage.

79. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors loosely coupled. Non shared storage.

80. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors loosely coupled. Shared storage.

81. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors loosely coupled. Minimized shared storage.

82. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Non shared storage.

83. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Shared storage.

84. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Minimized shared storage.

85. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors loosely coupled. Non shared storage.
86. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors loosely coupled. Shared storage.

87. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors loosely coupled. Minimized shared storage.

88. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Non shared storage.

89. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Shared storage.

90. P processors, and N nodes where $P = N \times 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Minimized shared storage.

B.5 P processors and N nodes where $P > N, P = N \times 2 + I$. (CLASS 5)

91. P processors and N nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Non shared storage.

92. P processors and N nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Shared storage.

93. P processors and N nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Shared storage.
node communications. Processors loosely coupled. Minimized shared storage.

94. $P$ processors and $N$ nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Non shared storage.

95. $P$ processors and $N$ nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Shared storage.

96. $P$ processors and $N$ nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Minimized shared storage.

97. $P$ processors and $N$ nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors loosely coupled. Non shared storage.

98. $P$ processors and $N$ nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors loosely coupled. Shared storage.


100. $P$ processors and $N$ nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors tightly coupled. Non shared storage.

101. $P$ processors and $N$ nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors tightly coupled. Shared storage.

102. $P$ processors and $N$ nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors tightly coupled. Minimized shared storage.

103. $P$ processors and $N$ nodes where $P > N$ and $P = N \times 2 + I$. Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors loosely coupled. Non shared storage.
104. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors loosely coupled. Shared storage.

105. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors loosely coupled. Minimized shared storage.

106. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Non shared storage.

107. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Shared storage.

108. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Minimized shared storage.

109. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Non shared storage.

110. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Shared storage.

111. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Minimized shared storage.

112. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Non shared storage.
113. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Shared storage.

114. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Minimized shared storage.

115. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors loosely coupled. Non shared storage.

116. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors loosely coupled. Shared storage.

117. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors loosely coupled. Minimized shared storage.

118. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Non shared storage.

119. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Shared storage.

120. P processors and N nodes where \( P > N \) and \( P = N \times 2 + I \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Minimized shared storage.


125. P processors and N nodes where $P > N$ and $P = N * 2 + I$. Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Shared storage.


B.6 P processors and N nodes, where $P > N$ and $P = N * I$ where $I > 2$. (CLASS 6)

127. P processors and N nodes, where $P > N$ and $P = N * I$ where $I > 2$. Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Non shared storage.
128. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Shared storage.

129. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors loosely coupled. Minimized shared storage.

130. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Non shared storage.

131. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications through node to node communications. Processors tightly coupled. Shared storage.

132. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors loosely coupled. Non shared storage.

133. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors loosely coupled. Shared storage.

134. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors tightly coupled. Non shared storage.

135. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors loosely coupled. Minimized shared storage.

136. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors tightly coupled. Non shared storage.

137. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host node. System supports communications by token ring through minimized paths. Processors tightly coupled. Shared storage.


140. P processors and N nodes, where $P > N$ and $P = N * I$ where $I > 2$. Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors loosely coupled. Shared storage.


142. P processors and N nodes, where $P > N$ and $P = N * I$ where $I > 2$. Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Shared storage.

143. P processors and N nodes, where $P > N$ and $P = N * I$ where $I > 2$. Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Shared storage.

144. P processors and N nodes, where $P > N$ and $P = N * I$ where $I > 2$. Queries are serviced through host node. System supports communications by token ring through multiple paths. Processors tightly coupled. Minimized shared storage.

145. P processors and N nodes, where $P > N$ and $P = N * I$ where $I > 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Non shared storage.

146. P processors and N nodes, where $P > N$ and $P = N * I$ where $I > 2$. Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Shared storage.

147. P processors and N nodes, where $P > N$ and $P = N * I$ where $I > 2$. Queries are serviced through host and requested node using semantic minimization. System
supports communications through node to node communications. Processors loosely coupled. Minimized shared storage.

148. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Non shared storage.

149. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Shared storage.

150. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Minimized shared storage.

151. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Non shared storage.

152. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Shared storage.

153. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors loosely coupled. Minimized shared storage.

154. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications through node to node communications. Processors tightly coupled. Non shared storage.

155. P processors and N nodes, where \( P > N \) and \( P = N \times I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System
supports communications by token ring through minimized paths. Processors tightly coupled. Shared storage.

156. P processors and N nodes, where \( P > N \) and \( P = N*I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through minimized paths. Processors tightly coupled. Minimized shared storage.

157. P processors and N nodes, where \( P > N \) and \( P = N*I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors loosely coupled. Non shared storage.

158. P processors and N nodes, where \( P > N \) and \( P = N*I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors loosely coupled. Shared storage.

159. P processors and N nodes, where \( P > N \) and \( P = N*I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Minimized shared storage.

160. P processors and N nodes, where \( P > N \) and \( P = N*I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Non shared storage.

161. P processors and N nodes, where \( P > N \) and \( P = N*I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Shared storage.

162. P processors and N nodes, where \( P > N \) and \( P = N*I \) where \( I > 2 \). Queries are serviced through host and requested node using semantic minimization. System supports communications by token ring through multiple paths. Processors tightly coupled. Minimized shared storage.
Glossary

Relational: Data storage format used to store and represent data in a relational database.

Tuple: The term used to refer to the collection of attributes in a row of a relational database, commonly compared to a record of a nonrelational structured file.

Attribute: The term used to refer to an individual entity of the tuple of a relational database, commonly compared to a field of a nonrelational structured file.

Fragmentation: The process of splitting the whole relational table into more manageable subtables. The process of fragmentation can be done against a table either horizontally (by groupings of tuples) or vertically (by groupings of attributes), or a combination of both.

Distributed Processing: The term used to describe the processing of work over several different processors either local or remote.

Data Integrity: The actions necessary to guarantee the correctness and availability of data.

Key: Attribute used to uniquely describe the tuple.
Bibliography

[Ancilotti, Bertolino, Fusani 89] Paolo Ancilotti, Antonia Bertolino, Mario Fusani 1989,
An Approach to efficient Distributed Transactions.
Distributed Computing. pages 201-212 volume 2.

[Apers 88] Peter Apers 1988,
Data Allocations in Distributed Database Systems.

[Borg, Blau, Graetch, Hermann 89] Anita Borg, Wolfgang Blau, Wolfgang Graetsch,
Ferdinand Hermann 1989,
Fault Tolerance under UNIX.

[Bracha, Toueg 87] Gabriel Bracha, Sam Toueg 1987,
Distributed Deadlock Detection.

[Davydov 91] Mark Davydov 1991,
Capacity Planning using DB2 index design.

[Farrag, Ozu 89] Abdel Aziz Farrag, M Tamer Ozu 1989,
Using Semantic Knowledge of Transactions to increase Concurrency.

Distributed FIFO Allocation of Identical Resources using Small Shared Space.
ACM Transactions on Programming Languages and Systems. pages 90-114 volume 11.
[Garcia-Molina, Barbara 85] Hector Garcia-Molina, Daniel Barbara 1985,
How to Assign Votes in a Distributed System.
Journal of the ACM. pages 841-860 volume 32.

[Gavish 90] Bezalel Gavish, Olivia R. Liu Sheng 1990,
Dynamic File Migration in Distributed Computer Systems.
Communications of the ACM. pages 177-189 volume 33.

[Glasgow, MacEwen 89] Janice I. Glasglow, Glenn H. MacEwen 1989,
An Operator Net model for Distributed Systems.
Distributed Computing. pages 196-209 volume 3.

[Kessels, Rem 90] Joep L. W. Kessels, Martin Rem 1990,
Designing Systolic, Distributed buffers with bounded response time.

[Lamport 86] Leslie Lamport 1986,
The Mutual Exclusion Problem: Part II Statement and Solutions.
Journal of the ACM. pages 327-348 volume 33.

[Lin 88] Huai-an Lin 1988,
A Methodology for constructing communication protocols with multiple concurrent Functions.
Distributed Computing. pages 23-40 volume 3.

[Maier, Ullman 83] David Maier, Jeffery Ullman 1983,
Fragmentations of Relations.

[Makki, Arndt 91] Kia Makki, Timothy Arndt 1991,
A Performance Measurement Study for Distributed Systems.

[Mattern 87] Friedmann Mattern 1987,
Algorithms for Distributed Termination Detection.
Distributed Computing pages 161-175 volume 2.

[Pirahesh, Mohan, Cheng, Liu, Selinger 90] Hamid Pirahesh, C. Mohan, Josephine Cheng, T. S. Liu, Pat Selinger 1990,
DPPS proceedings prior to release of publication.
[Rabin 89] Micheal O. Rabin 1989,
Journal of the ACM. pages 335-348 volume 36.

[Sacca, Wederhold 85] Domenico Sacca, Gio Wederhold 1985,
Database Partitioning in a Cluster of Processors.
ACM Transactions on Database Systems. pages 29-56 volume 10.

[Snyder 89] Micheal Snyder 1989,
DB2 Retrieval Efficiency.

[Ullman 82 ] Jeffery Ullman 1982,
Principles of Database Systems.

[Werman 91] Aaron Werman 1991,
How many DB2 subsystems are needed?.

[Werman 90] Aaron Werman 1990,
Mostly DB2.

[Wong, Katz 89] E. Wong, R. H. Katz 1989,
Distributing a Database for Parallelism.
ACM Transactions on Database Systems. pages 23-29 volume 5.

[Valduriez, Khoshafian 90] atrick Valduriez, Setrag Knoshafian 1990,
Parallel Evaluation of the Database Relation.