



# VCU

Virginia Commonwealth University  
**VCU Scholars Compass**

---

Theses and Dissertations

Graduate School

---

1995

## A generalized system performance model for object-oriented database applications

Ellen Moore Walk

Follow this and additional works at: <https://scholarscompass.vcu.edu/etd>



Part of the [Business Commons](#)

© The Author

---


Downloaded from

<https://scholarscompass.vcu.edu/etd/5613>

This Dissertation is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact [libcompass@vcu.edu](mailto:libcompass@vcu.edu).

School of Business  
Virginia Commonwealth University

This is to certify that the dissertation prepared by Ellen Moore Walk entitled A Generalized System Performance Model for Object-Oriented Database Applications has been approved by this committee as satisfactory completion of the dissertation requirements for the degree of Doctor of Philosophy in Business.



Dr. Richard T. Redmond, Director

School of Business



Dr. F. Paul Fuhs, Committee Member

School of Business



Dr. Richard J. Coppins, Committee Member

School of Business



Dr. Robert L. Andrews, Committee Member

School of Business



Dr. Lawrence W. West, Committee Member

Department of  
Mathematical Sciences



Dr. Howard P. Tuckman, Dean

School of Business

December 14, 1995

Date

©Ellen Moore Walk 1995  
All Rights Reserved

A GENERALIZED SYSTEM PERFORMANCE MODEL  
FOR OBJECT-ORIENTED DATABASE APPLICATIONS

A Dissertation Submitted in Partial Fulfillment of the Requirements  
for the Degree of Doctor of Philosophy in the  
Department of Information Systems  
School of Business  
Virginia Commonwealth University

By  
Ellen Moore Walk

Master of Business Administration, University of Richmond, 1983  
Bachelor of Science, College of William & Mary, 1977

Director: Richard T. Redmond, D.B.A.  
Department of Information Systems

Virginia Commonwealth University  
Richmond, Virginia  
December, 1995

## ACKNOWLEDGEMENTS

I would like to thank the members of my committee, Dr. Rich Redmond, Dr. Paul Fuhs, Dr. Rich Coppins, Dr. Bob Andrews, and Dr. Larry West, for their professional guidance and encouragement during the development of this dissertation. I will always be grateful for their friendship as well as the example they set in teaching and research.

## DEDICATION

This dissertation is dedicated to my husband, John, and my children, Sarah and John.

## TABLE OF CONTENTS

	Page
Acknowledgements	ii
Dedication	iii
List of Tables	vii
List of Figures	ix
Abstract	xiv
Chapter	
I. INTRODUCTION.....	1
Differences Between Relational and Object-Oriented Database Applications .....	1
The Significance of Performance in Object-Oriented Databases .....	2
The Need for Predicting Application Performance.....	3
Research Questions.....	4
II. LITERATURE REVIEW .....	5
Object-Oriented Database Concepts.....	5
Benchmarking Approaches to Modeling OODB Performance.....	6
The OO1 Benchmark.....	6
The HyperModel Benchmark.....	9
The OO7 Benchmark.....	11
Using the Current Study to Extend Benchmark Studies.....	17
Simulation Approaches to Modeling OODB Performance .....	19
Analyzing Processes to Identify Sources of Variability .....	20
Software Metrics for Object-Oriented Applications.....	21
III. RESEARCH METHOD .....	25
Objectives of Study.....	25

Operating Platform .....	25
Test Instrument .....	26
Overview of Experimental Design .....	26
Variables Held Constant for All Three Experiments .....	26
Overview of the OO7 Operations Selected for This Study .....	27
Experimental Design for Research Question 1: Aggregation .....	28
Review of Research Question on Aggregation .....	28
Definition of Terms .....	28
Research Hypothesis for Aggregation .....	29
Operational Definitions: Dependent Variable .....	29
Operational Definitions: Independent Variable .....	30
Manipulation of the Independent Variable .....	30
Methodology for Data Collection .....	32
Data Analysis: Model and Statistical Hypothesis .....	33
Experimental Design for Research Question 2: Inheritance .....	34
Review of Research Question on Inheritance .....	34
Definition of Terms .....	34
Research Hypotheses for Inheritance .....	35
Operational Definitions: Dependent Variable .....	36
Operational Definitions: Independent Variables .....	36
Manipulation of the Independent Variables .....	36
Methodology for Data Collection .....	40
Data Analysis: Model and Statistical Hypotheses .....	43
Experimental Design for Research Question 3: Workload Sequence .....	44
Review of Research Question on Workload Sequence .....	44
Definition of Terms .....	44
Research Hypotheses for Workload Sequence .....	45
Operational Definitions: Dependent Variable .....	46
Operational Definitions: Independent Variables .....	46
Methodology for Data Collection .....	50
Data Analysis: Model and Statistical Hypotheses .....	52
 IV. RESULTS .....	 54
Results for Research Question 1: Aggregation .....	54
Results for Research Question 2: Inheritance .....	64
Treatment 1 vs. Treatment 2 (Effect of Class Definitions) .....	64
Treatment 2 vs. Treatment 3 (Effect of Run-Time Polymorphism) .....	65
Results for Research Question 3: Workload Sequence .....	69
Effect of # Workload Units Processed .....	69
Effect of Dummy Variables Indicating Preceding Conditions .....	69
Further Analysis of Variability in Response Times .....	70



Effect of the Seed Used to Generate Random Operations Stream.....	71
Effect of Removing Q8 and T1 From the Workload.....	71
<b>V. CONCLUSIONS .....</b>	<b>92</b>
Summary of Results .....	92
Effect of Degree of Aggregation on Response Time .....	92
Effect of Degree of Inheritance on Response Time .....	93
Effect of Sequence of Operations on Response Time .....	93
Implications for Applications Designers.....	94
Contributions and Limitations of Study .....	95
Future Work.....	96
<b>BIBLIOGRAPHY .....</b>	<b>97</b>
<b>APPENDICES</b>	
A Program for Generating Database Operations in Pseudorandom Order .....	101
B OO7 Benchmark, Modified for Simulated Transaction Workload.....	104
C Modifying Degree of Inheritance in OO7 Class Definitions .....	123
D Modifying Degree of Inheritance in OO7 Traversal Operations to .....	131
Use Three Levels of Run-Time Polymorphism	
E Response Time Histograms and Time-Series Charts for Database 7/200/3/3 .....	140
F Response Times Using Different Seeds for Random Number Generator and .....	149
Deleting Q8 and T1 From Workload	
<b>VITA .....</b>	<b>158</b>

## LIST OF TABLES

Table 2.1	OO7 Benchmark Database Parameters	14
Table 2.2	OO7 Benchmark Database Operations	15
Table 3.1	Overview of OO7 Operations Selected for This Study	27
Table 3.2	Levels Chosen for Independent Variable, #Hookups Processed	32
Table 3.3	Manipulation of the Independent Variables to Test the Effects of Inheritance	40
Table 3.4	Levels Chosen for Independent Variable, #Workload Units	48
Table 3.5	Distribution of Database Operations	51
Table 4.1	Q4 Aggregation Model	56
Table 4.2	Q5 Aggregation Model	57
Table 4.3	Q6 Aggregation Model	58
Table 4.4	T6 Aggregation Model	59
Table 4.5	T1 Aggregation Model, All Databases Combined	60
Table 4.6	T1 Aggregation Model, Databases with 100 Atomic Parts Per Composite Part	61
Table 4.7	T1 Aggregation Model, Databases with 150 Atomic Parts Per Composite Part	62
Table 4.8	Analysis of ObjectStore Physical Segments Storing the OO7 Database	63
Table 4.9	Comparison of Database Object Counts and Size	64

Table 4.10	ANOVA for Q6	66
Table 4.11	ANOVA for T6	67
Table 4.12	ANOVA for T1	68
Table 4.13	Q1 Workload Regression Model	73
Table 4.14	Q2 Workload Regression Model	74
Table 4.15	Q3 Workload Regression Model	75
Table 4.16	Q4 Workload Regression Model	76
Table 4.17	Q5 Workload Regression Model	77
Table 4.18	Q6 Workload Regression Model	78
Table 4.19	Q8 Workload Regression Model	79
Table 4.20	T1 Workload Regression Model	80
Table 4.21	T6 Workload Regression Model	81

## LIST OF FIGURES

Figure 2.1	OO7 Object Model Original Implementation with One Level of Inheritance (Assembly)	12
Figure 2.2	OO7 “Assembly Hierarchy” Two-Dimensional Model of Object Instances	13
Figure 3.1	OO7 Object Model Modified to Add a Second Level of Inheritance (DesignObject)	38
Figure 3.2	OO7 Object Model With Third Level of Inheritance (RootObject)	39
Figure 4.1	Frequency Distribution for Q1 Response Times, Database 4/100/3/3	82
Figure 4.2	Q1 Response Times, In Order of Occurrence in Workload, Database 4/100/3/3	82
Figure 4.3	Frequency Distribution for Q2 Response Times, Database 4/100/3/3	83
Figure 4.4	Q2 Response Times, In Order of Occurrence in Workload, Database 4/100/3/3	83
Figure 4.5	Frequency Distribution for Q3 Response Times, Database 4/100/3/3	84
Figure 4.6	Q3 Response Times, In Order of Occurrence in Workload, Database 4/100/3/3	84
Figure 4.7	Frequency Distribution for Q4 Response Times, Database 4/100/3/3	85
Figure 4.8	Q4 Response Times, In Order of Occurrence in Workload, Database 4/100/3/3	85
Figure 4.9	Frequency Distribution for Q5 Response Times, Database	86

Figure 4.10	Q5 Response Times, In Order of Occurrence in Workload, Database 4/100/3/3	86
Figure 4.11	Frequency Distribution for Q6 Response Times, Database 4/100/3/3	87
Figure 4.12	Q6 Response Times, In Order of Occurrence in Workload, Database 4/100/3/3	87
Figure 4.13	Frequency Distribution for Q8 Response Times, Database 4/100/3/3	88
Figure 4.14	Q8 Response Times, In Order of Occurrence in Workload, Database 4/100/3/3	88
Figure 4.15	Frequency Distribution for T1 Response Times, Database 4/100/3/3	89
Figure 4.16	T1 Response Times, In Order of Occurrence in Workload, Database 4/100/3/3	89
Figure 4.17	Frequency Distribution for T6 Response Times, Database 4/100/3/3	90
Figure 4.18	T6 Response Times, In Order of Occurrence in Workload, Database 4/100/3/3	90
Appen. E.1	Frequency Distribution for Q1 Response Times, Database 7/200/3/3	140
Appen. E.2	Q1 Response Times, In Order of Occurrence in Workload, Database 7/200/3/3	140
Appen. E.3	Frequency Distribution for Q2 Response Times, Database 7/200/3/3	141
Appen. E.4	Q2 Response Times, In Order of Occurrence in Workload, Database 7/200/3/3	141
Appen. E.5	Frequency Distribution for Q3 Response Times, Database 7/200/3/3	142

Appen. E.6	Q3 Response Times, In Order of Occurrence in Workload, Database 7/200/3/3	142
Appen. E.7	Frequency Distribution for Q4 Response Times, Database 7/200/3/3	143
Appen. E.8	Q4 Response Times, In Order of Occurrence in Workload, Database 7/200/3/3	143
Appen. E.9	Frequency Distribution for Q5 Response Times, Database 7/200/3/3	144
Appen. E.10	Q5 Response Times, In Order of Occurrence in Workload, Database 7/200/3/3	144
Appen. E.11	Frequency Distribution for Q6 Response Times, Database 7/200/3/3	145
Appen. E.12	Q6 Response Times, In Order of Occurrence in Workload, Database 7/200/3/3	145
Appen. E.13	Frequency Distribution for Q8 Response Times, Database 7/200/3/3	146
Appen. E.14	Q8 Response Times, In Order of Occurrence in Workload, Database 7/200/3/3	146
Appen. E.15	Frequency Distribution for T1 Response Times, Database 7/200/3/3	147
Appen. E.16	T1 Response Times, In Order of Occurrence in Workload, Database 7/200/3/3	147
Appen. E.17	Frequency Distribution for T6 Response Times, Database 7/200/3/3	148
Appen. E.18	T6 Response Times, In Order of Occurrence in Workload, Database 7/200/3/3	148
Appen. F.1	Q1 Response Times, Database 4/100/3/3 Using Different Seeds for Random Number Generator	149

Appen. F.2	Q1 Response Times, Deleting Q8 and T1 From Workload Database 4/100/3/3	149
Appen. F.3	Q2 Response Times, Database 4/100/3/3 Using Different Seeds for Random Number Generator	150
Appen. F.4	Q2 Response Times, Deleting Q8 and T1 From Workload Database 4/100/3/3	150
Appen. F.5	Q3 Response Times, Database 4/100/3/3 Using Different Seeds for Random Number Generator	151
Appen. F.6	Q3 Response Times, Deleting Q8 and T1 From Workload Database 4/100/3/3	151
Appen. F.7	Q4 Response Times, Database 4/100/3/3 Using Different Seeds for Random Number Generator	152
Appen. F.8	Q4 Response Times, Deleting Q8 and T1 From Workload Database 4/100/3/3	152
Appen. F.9	Q5 Response Times, Database 4/100/3/3 Using Different Seeds for Random Number Generator	153
Appen. F.10	Q5 Response Times, Deleting Q8 and T1 From Workload Database 4/100/3/3	153
Appen. F.11	Q6 Response Times, Database 4/100/3/3 Using Different Seeds for Random Number Generator	154
Appen. F.12	Q6 Response Times, Deleting Q8 and T1 From Workload Database 4/100/3/3	154
Appen. F.13	Q8 Response Times, Database 4/100/3/3 Using Different Seeds for Random Number Generator	155
Appen. F.14	T1 Response Times, Database 4/100/3/3 Using Different Seeds for Random Number Generator	156
Appen. F.15	T1 Response Times, Deleting Q8 and T1 From Workload Database 4/100/3/3	156

Appen. F.16	T6 Response Times, Database 4/100/3/3 Using Different Seeds for Random Number Generator	157
Appen. F.17	T6 Response Times, Deleting Q8 and T1 From Workload Database 4/100/3/3	157



## ABSTRACT

A GENERALIZED SYSTEM PERFORMANCE MODEL FOR OBJECT-ORIENTED  
DATABASE APPLICATIONS

By Ellen Moore Walk, Ph.D.

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 1995

Major Director: Dr. Richard T. Redmond, Department of Information Systems

Although relational database systems have met many needs in traditional business applications, such technology is inadequate for non-traditional applications such as computer-aided design, computer-aided software engineering, and knowledge bases. Object-oriented database systems (OODB) enhance the data modeling power and performance of database management systems for these applications.

Response time is an important issue facing OODB. However, standard measures of on-line transaction processing are irrelevant for OODB. Benchmarks compare alternative implementations of OODB system software, running a constant application workload. Few attempts have been made to characterize performance implications of OODB application design, given a fixed OODB and operating system platform.

In this study, design features of the OO7 Benchmark database application (Carey, DeWitt, and Naughton, 1993) were varied to explore the impact on response time to perform database operations. Sensitivity to the degree of aggregation and to the degree of inheritance in the application were measured. Variability in response times also was measured, using a sequence of database operations to simulate a user transaction workload.

Degree of aggregation was defined as the number of relationship objects processed during a database operation. Response time was linear with the degree of aggregation. The size of the database segment processed, compared to the size of available memory, affected the coefficients of the regression line.

Degree of inheritance was defined as the Number of Children (Chidamber and Kemerer, 1994) in the application class definitions, and as the extent to which run-time polymorphism was implemented. In this study, increased inheritance caused a statistically significant increase in response time for the OO7 Traversal1 only, although this difference was not meaningful.

In the simulated transaction workload of nine OO7 operations, response times were highly variable. Response times per operation depended on the number of objects processed and the effect of preceding operations on memory contents. Operations that used disparate physical segments or had large working sets relative to the size of memory caused large increases in response time. Average response times and variability were

reduced by removing these operations from the sequence (equivalent to scheduling these transactions at some time when the impact would be minimized).

## CHAPTER I

### INTRODUCTION

#### DIFFERENCES BETWEEN RELATIONAL AND OBJECT-ORIENTED DATABASE APPLICATIONS

Although relational database systems have met many needs of traditional business data processing applications, such technology has proven inadequate for non-traditional applications such as computer-aided design, manufacturing, and software engineering (CAD/CAM/CASE), office information systems, and knowledge bases. These non-traditional applications are characterized by large, complex data structures, nested and interrelated data, and multiple versions of database entities. CAD, CAM, and CASE applications, sometimes classified as "design applications" [Hurson et al, 1993; Maier, 1989], typically require the processing of components at different levels of abstraction (equivalent objects). An example of this is processing VLSI CAD data at the gate, component, and chip levels. A transaction in such an application can have a much longer duration than traditional database transactions, too, as objects may be "checked out" by designers.

Relational database technology, designed to process records made up of relatively simple, fixed-format data, does not currently provide the functionality and performance to meet all needs of such advanced applications. Many existing design applications have resorted to the use of custom programming on top of file management systems to obtain

reasonable performance, avoiding use of a database management system altogether. [Cattell, 1994; Hurson et al, 1993].

Object-oriented database systems (OODB) have been developed as one of several approaches to enhancing the data modeling power and performance of database management systems (DBMS) for advanced applications. (Other approaches under research are extended relational, functional, and semantic databases [Cattell, 1994].) Object-oriented database systems have the capability of improved performance due primarily to (1) the ability to represent relationships among data by using unique object identifiers or pointers, and (2) the use of object-oriented programming languages with database features. First, object identifiers or virtual pointers can be translated very quickly into the physical memory addresses of related objects. By comparison, relational queries on attributes of a relation involve an extra associative mapping step to translate attribute values into addresses. Second, using a single language for application program code as well as database code overcomes the "impedance mismatch" exhibited in relational systems, and opens up possibilities for the use of more efficient data structures tailored to the application. Both of these features also reduce the number of procedure calls during program execution, thereby reducing overhead. [Maier, 1989]

## THE SIGNIFICANCE OF PERFORMANCE IN OBJECT-ORIENTED DATABASES

System performance (primarily response time) is considered by some to be the most important issue facing OODB, more than functionality [Hurson et al, 1993; Cattell and Skeen, 1992; Cattell, 1994; Maier, 1989]. However, non-traditional applications differ so much from traditional business on-line transaction processing that the standard measures of database management system performance are inadequate. These include the

Wisconsin and the TPC-A benchmarks [Gray, 1991; Anon. et al, 1985]. It is possible that many business database applications today, if they were to be modeled and implemented using the enhanced semantic representations of object-oriented design, would also require more complex processing than envisioned by these standard benchmarks.

Because of the uniqueness of non-traditional applications, and the lack of data on actual OODB implementations, several attempts have been made to develop general tools for measuring OODB performance. Key studies include the OO7 benchmark [Carey et al, 1993], preceded by the OO1 benchmark [Cattell and Skeen, 1992; Rubenstein et al, 1987] and the HyperModel benchmark [Berre and Anderson, 1991; Anderson et al, 1990]. The objective of these benchmarks is to run a constant, artificial application workload across different DBMS architectures (including relational as well as OODB).

## THE NEED FOR PREDICTING APPLICATION PERFORMANCE

An important use of performance modeling during application development is during the design phase, to estimate the future performance of a system. Hardware requirements (particularly memory), application data structures and database access methods, and network traffic are key issues. It is important to know what application characteristics and system architecture factors will have the most impact on the ultimate performance of the system. It is also important to be able to compare alternative designs conveniently during the design stage, on the basis of the critical design parameters, before the system is implemented.

Synthetic benchmarks are only helpful to the extent they characterize the workload of the actual application being developed, and can be run on an existing platform. Detailed simulation models generally are not cost-effective during design and

development, when details of the future system may be sketchy and subject to error. However, simple models including the primary parameters of interest could be useful, to reduce the risk of poor performance in the final system.

Workload characterization of an existing system normally provides the basis for a realistic synthetic workload, or for representative parameters in an analytic or simulation model of a system [Heidelberger and Lavenberg, 1984, Ferrari et al, 1983]. A study of the impact of application software factors on performance, given hardware, operating system, and database software, would extend the usefulness of previously published benchmarks.

## RESEARCH QUESTIONS

For a given platform (hardware, operating system, and commercial OODB):

- 1) What is the relationship between the degree of aggregation in an OODB user application and the response time to do database operations?
- 2) What effect does degree of inheritance in a user application have on response time?
- 3) How does the sequence of database operations in a user transaction workload influence response time variability?

## CHAPTER II

### LITERATURE REVIEW

#### OBJECT-ORIENTED DATABASE CONCEPTS

An informative, comprehensive introduction to object-oriented databases can be found in both [Cattell, 1994] and [Kim, 1990]. Object-oriented data modeling and databases are extensions of the object-oriented programming paradigm. An **object** can be defined as a real-world or abstract entity, (e.g., documents, software modules, equipment parts, people) about which data will be stored. Objects have behaviors--functions or operations done by or to an object; these operations are often called **methods**. Abstract data types, which encapsulate data and functions/operations on that data, are known as **classes** or **types** in object-oriented programming languages.

Object types can be related to each other in a **type hierarchy** of supertypes and subtypes. Subtypes can **inherit** commonly-held attributes and operations from supertypes, and have unique attributes and operations of their own. Subtypes have an "is-a" relationship with their supertype(s).

The motivation behind OODB is the need to model and implement complex objects which are not adequately handled by the relational approach. For instance, objects may be composed of other objects (**aggregation**). Aggregation relationships between objects are also called "part-of" relationships. Also, objects or their attributes may be very large (e.g., a bitmap or text), rather than being comprised of simple data types (integers, characters, etc.). There may be multiple versions of objects to be stored simultaneously in



the database. Often complex objects need to be stored and processed as ordered sets or lists, which are difficult to represent in relational databases.

OODB architecturally combine database capabilities with object-oriented programming languages, e.g., C++ and Smalltalk. There is a single language for database and programming operations, and a unified approach for extending object-oriented modeling to programming and database implementation. Logical relationships between objects can be matched in the physical implementation, as opposed to requiring the joining of related data at run time, again promoting uniformity between logical and physical designs.

## BENCHMARKING APPROACHES TO MODELING OODB PERFORMANCE

Since 1987, several benchmarks for OODB have been designed by database researchers. These are described below in order of chronological development. (An annotated bibliography of published papers, technical reports and theses on object database performance and benchmarks is provided in [Chaudhri, 1995]).

### The OO1 Benchmark

With the proposal of the first benchmark for object-oriented applications, [Rubenstein et al, 1987] at Sun Microsystems began a series of publications describing the evolution of the "Sun Benchmark" into the OO1 Benchmark [Cattell, 1988; Gray, 1991; Cattell and Skeen, 1992]. This work was motivated by the inadequacy of traditional DBMS benchmarks for representing engineering applications on workstations. The objective was to measure the total response time to run a given workload, with operations

and data more typical of engineering applications, across different types of DBMS (relational, object-oriented, network, and hierarchical). The original benchmark application database consisted of three record types: person, document, and authors relating persons and documents. Seven simple operations were measured: name lookup, range lookup, group lookup, reference lookup, record insert, sequential scan, and database open.

In [Cattell, 1988] the author/document database was replaced with a database of parts on a circuit board and connections between them, responding to criticism [e.g., Duhl and Damon, 1988] that the original database did not adequately represent object-oriented features. Also, the seven operations were reduced to three generic operations by 1) grouping name and range lookup into a lookup operation, 2) replacing reference and group lookup with a traversal measure, and 3) keeping the insert operation but eliminating sequential scan and database open from the benchmark.

The most recent version of this approach was the Object Operations Benchmark (OO1), [Cattell and Skeen, 1992; also published in Gray, 1991]. Cattell and Skeen kept the focus on three generic operations: object lookup, traversal of connections between objects, and inserting objects. These were thought to be the most frequently occurring operations in engineering applications, based on feedback on the earlier papers and on interviews with engineers using CASE and CAD. The parts database was kept, consisting of two logical records with data on parts and connections. Attributes of these records were simple data types such as strings and integers. The database had a somewhat regular structure, with exactly three connections from each part to other randomly selected parts, most of which were close in "locality" (defined as having numerically close part ids). Each part was thus be reachable from a random number of parts.

[Cattell and Skeen, 1992] ran this benchmark across several DBMS platforms. The overall size of the parts database was varied experimentally with two levels: one database "small" enough to fit in main memory, and the other "large" (scaled up by a factor of ten) enough to require access to secondary storage. The two parts databases thus differed in the size of their working set. However, the various DBMS platforms tested using the benchmark differed in numerous respects which were **not** controlled experimentally in their study. The systems studied varied in terms of:

- parts database size after implementation (e.g., "small" parts database varied 10 - 50% in size across DBMS platforms)
- cache size
- cache algorithm
- page size
- means of mapping part id to disk address (hash index vs. B-tree vs. link structure; authors chose whatever was the fastest available for each DBMS)
- concurrency control mechanisms and other overhead functions
- binding of DBMS with benchmark application code vs. interprocess calls

For example, one of the relational databases had a much longer total response time, due to confounding effects of cache differences, the overhead for DBMS calls, and indexing rather than linked structures for traversals. The authors concluded that all three factors--binding of DBMS code with application code, caching large amounts of the database in main memory, and providing new access methods--were important.

The Object Operations Benchmark paper is significant to the proposed research study in the identification of potential key factors affecting performance of object-oriented database systems, e.g., cache size vs. working set, access methods. However, it leaves open the opportunity to study the effect of varying additional factors under experimental control. In addition, because application database characteristics are not varied in this

benchmark, there is a need to determine relationships and interactions of the application workload with hardware and DBMS variables. Finally, because there are no intermediate measures of response time per operation under a realistic mix of operations, this study does not look at the variability in performance, only total time to run the benchmark for each individual operation type.

### The HyperModel Benchmark

The HyperModel benchmark, as described in [Berre and Anderson, 1991; more detail in Anderson et al, 1990] was also an attempt to develop a generic application to be used to evaluate performance across any DBMS. General DBMS requirements for engineering applications were identified by the authors with an in-house survey of such applications at Tektronix, Inc., e.g., modeling of complex object structures, description of different data types, integration with application programming languages. Engineering applications were not characterized along any quantitative dimensions.

These authors criticized the OO1 benchmark on the basis that the generic application database was too simple to measure transitive closures and other traversal operations. In this work, a hypertext application database was used to model more complex objects and additional operations more representative of engineering applications such as CAD and CASE. The HyperModel database organized information about textual documents and the relationships between parts of a document. While the OO1 benchmark ultimately evolved into the measurement of three significant operation types, the HyperModel benchmark expanded on the seven original operations from the Sun Benchmark. The HyperModel Benchmark measured 20 comprehensive operations, grouped into the following 10 categories, against the hypothetical database:

- Name Lookup
- Range Lookup
- Group Lookup
- Reference Lookup
- Sequential Scan
- Closure Traversal
- Closure Operations
- Editing
- Create-and Delete
- Open-and-Close

To obtain their performance measurements, each of the 20 operations was run 50 trials in a row; i.e., operations were not mixed. To measure cold-start results (without caching), the database was closed before each new trial, and each trial was run on 50 different randomly selected objects. The total time for the cold run was then divided by 50 to get the mean time for that operation (in milliseconds/operation). To measure results with caching, the same operation was run again on the same 50 objects, and the mean time for that operation under warm-start conditions was calculated.

No other metrics were collected while running this benchmark. In the conclusion of [Anderson et al, 1990], this lack of descriptive data was recognized, and one could question how this and other benchmarks could be used to predict the performance of a particular application or mix of applications. The HyperModel database had a very regular structure of relationships between nodes, and little variability in object size. While it had a richer structure than the OO1 parts database, it was not known whether such uniformity would be characteristic of most applications with complex objects. Running the database operations one at a time, as opposed to mixing operations in a more realistic workload, would understate variability that would be relevant in real applications.

### The OO7 Benchmark

This benchmark, developed by Michael Carey, David DeWitt, Jeffrey Naughton and others at the University of Wisconsin, has superseded the Cattell OO1 benchmark because of its richness and flexibility, and its reproducibility. Evolution of the benchmark is described in [Carey et al, 1993, Carey et al, 1994] and additional technical reports from the University of Wisconsin.

As with OO1, the purpose of OO7 was to provide a controlled user workload across multiple OODB platforms. In the 1993 Wisconsin study, each platform consisted of a different OODB product and roughly comparable hardware. (There were also operating system differences across the platforms that heavily influenced some of the results, and these were noted in the technical report where applicable.) The purpose was to gain insight into the design of the OODB system software, given a fairly representative, constant user workload.

OO7 used a single application database, designed to be like CAD or CASE database. Figures 2.1 and 2.2 illustrate (1) the object class model for the OO7 database, following the diagramming conventions of [Rumbaugh et al, 1991], and (2) a two-dimensional representation of the OO7 database as it was constructed in memory.

Figure 2.1  
OO7 OBJECT MODEL  
Original Implementation with One Level of Inheritance (Assembly)

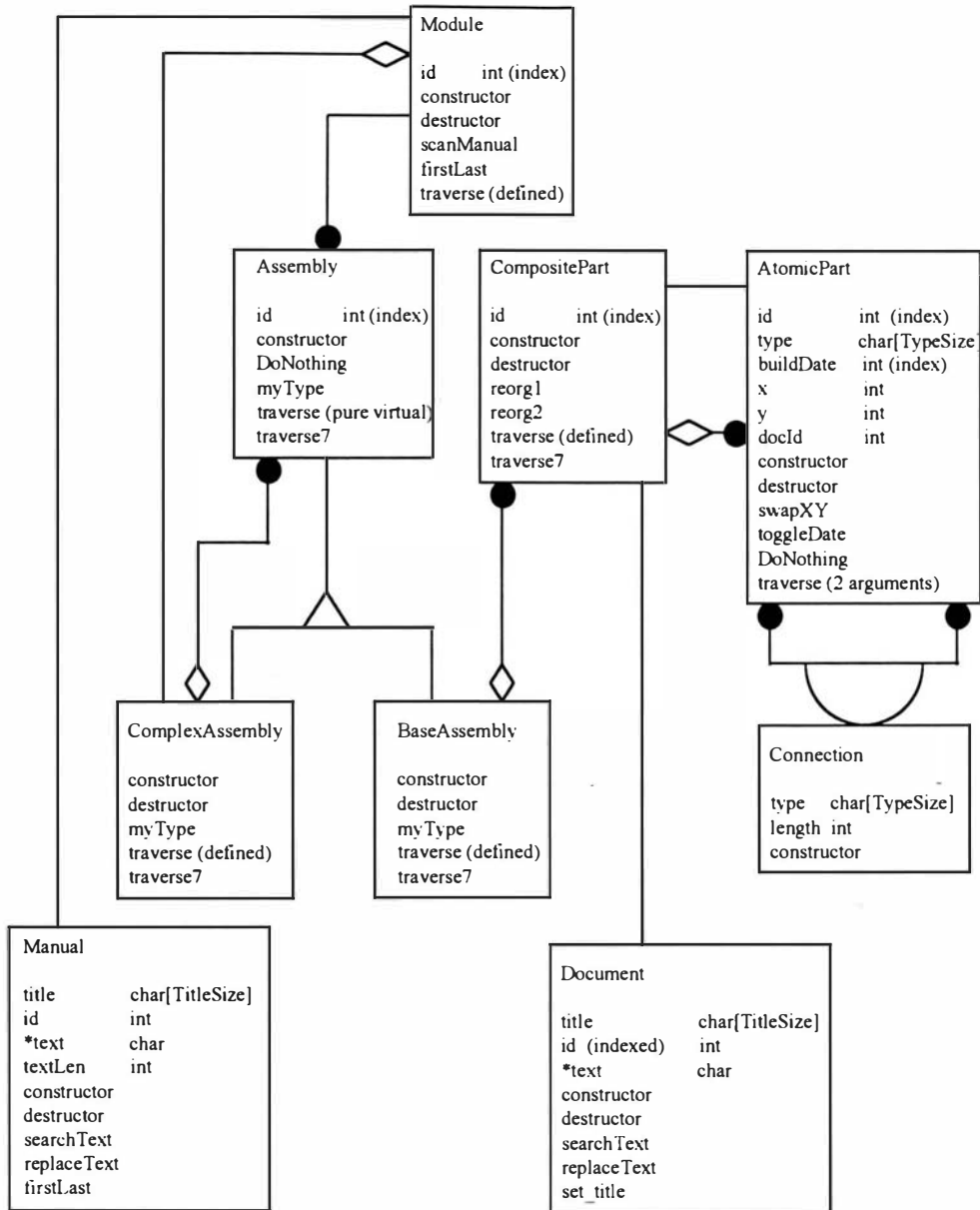
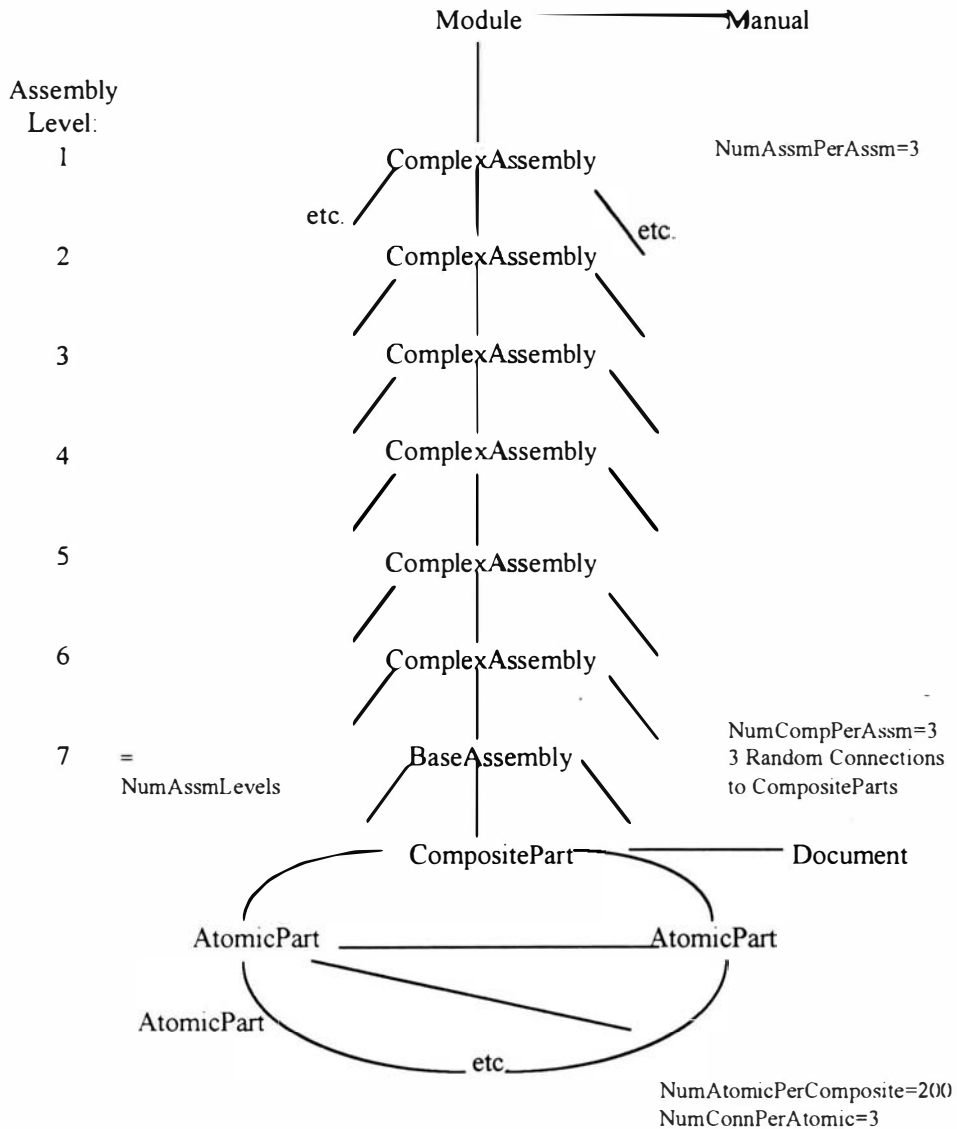


Figure 2.2  
 OO7 “Assembly Hierarchy”  
 Two-Dimensional Model of Object Instances





In the OO7 benchmark, database size was varied, but several other application parameters were held constant, as summarized in Table 2.1:

PARAMETER	SMALL	MEDIUM	LARGE
Number of Modules	1	1	10
Document Size (bytes)	2,000	20,000	20,000
Manual Size (bytes)	100 K	1 M	1 M
Number of Composites Per Module	500	500	500
Number of Assemblies Per Assembly	3	3	3
Number of Composites Per Base	3	3	3
Number of Connections Per Atomic	3,6,9	3,6,9	3,6,9
Number of Assembly Levels	7	7	7
Number of Atomic Per Composite	20	200	200

Multiple types of traversals, queries, insert, delete, reorganizations were run on the database. These operations were each executed once and the response time was measured. One or multiple repetitions of the same operation could be executed per database transaction, but different operations were not mixed in a simulated sequence/script. A summary of operations in the OO7 benchmark is found in Table 2.2.

TABLE 2.2  
OO7 BENCHMARK DATABASE OPERATIONS  
(Carey et al, 1993)

DELETE	Delete 10 randomly chosen composite parts.
INSERT	Create 10 new composite parts, and the set of atomic parts for each. Add a reference to each new composite part from a randomly chosen base assembly.
QUERY1	Randomly choose 10 atomic parts by lookup on their id field. An index on atomic parts can be used. (Exact Match Lookup)
QUERY2	Selection of atomic parts with buildDates in the most recent 1% of the range of possible buildDates. A B+ tree index is used. (Range query)
QUERY3	Selection of atomic parts with buildDates in the most recent 10% of the range of possible buildDates. A B+ tree index is used. (Range query)
QUERY4	Randomly choose 10 document objects by lookup on their title field, and find all base assemblies that use the composite part corresponding to the document. An index on document can be used. (Path lookup--document title->document id->corresponding composite part->corresponding base assembly)
QUERY5	Single-level make. Find all base assemblies rendered out of date by using a composite part with a more recent build date.
QUERY6	Find all assemblies (base or complex) that use (directly or transitively) a composite part with a more recent build date than the assembly's build date.
QUERY7	Iterate through all atomic parts. Checks scan speed. (Duplicates some other queries that do not use an index.)
QUERY8	Value join between documents and atomic parts on document id and atomic part id. (Key to Foreign Key Join)

TABLE 2.2  
(Continued)

REORGANIZATION1	Visit all composite parts, deleting and then newly inserting 1/2 of each composite part's atomic parts graph.
REORGANIZATION2	Randomly delete and reinsert as many atomic parts per composite part as each one has, testing the deletion/reallocation handling of the system.
TRAVERSAL1	Dense traversal. Visit all assemblies and atomic parts, do no work.
TRAVERSAL2a	Dense traversal with sparse updates. Visit all atomic parts, update one atomic part per composite part.
TRAVERSAL2b	Dense traversal with dense updates. Visit all atomic parts, update every atomic part as it is encountered.
TRAVERSAL2c	Dense traversal with dense, repeated updates. Visit all atomic parts, update each part 4 times.
TRAVERSAL3a	Traversal 2a with update to indexed field.
TRAVERSAL3b	Traversal 2b with update to indexed field.
TRAVERSAL3c	Traversal 2c with update to indexed field.
TRAVERSAL6	Sparse traversal: visit each base assembly, and each of the composite parts per base assembly. For each composite part, visit only the root atomic part.
TRAVERSAL8	Scan the manual object.
TRAVERSAL9	Looks at the beginning and ending characters of the manual object.

NOTE: TRAVERSALS 4, 5, and 7 of the benchmark were run, but provided no added insight in the University of Wisconsin tests.

The OO7 benchmark has been used as a standard workload to test various aspects of OODB performance. In [White and DeWitt, 1995], for example, OO7 was used to compare alternative crash recovery techniques in QuickStore, a memory-mapped store built on top of the Exodus storage manager. In this research, the focus was on the subset of OO7 operations that perform database updates. In [Carey et al, 1994] future plans were described to extend the benchmark to provide a customizable, multiuser workload. Others have used the benchmark to experiment with fetch policies and program optimization techniques [Brownsmith, 1995]. The benchmark also provides a useful training model for teaching OODB in the classroom and in the laboratory.

## USING THE CURRENT STUDY TO EXTEND BENCHMARK STUDIES

However, from the point of view of application developers designing database applications to run on an OODB, the question is: what various application dimensions affect performance, given a single OODB platform? This is the inverse of the question answered by a benchmark, but it is the relevant question faced by corporate and government developers after they have selected a platform and OODB vendor, and are designing a variety of application systems.

Taking this latter perspective, the OO7 benchmark also can be used as a valid starting point. The purpose of this study was to continue to examine the design/performance space in OODB applications. This was done by varying static,

structural parameters in the user database, and by simulating a dynamic transaction workload against this database.

This added to the current body of knowledge on ODBMS performance in the following ways. First, the impact of variability in database structural features on database response time was measured, and an attempt was made to derive a statistical relationship between them. Second, variance in response time for database operations was studied, to simulate the variance users would experience under different conditions.

Individual application databases can differ in their static structural features along the following dimensions:

- database size
- interconnectivity relationships between objects
- degree of aggregation
- degree of inheritance
- complexity of data members/structures within classes
- use of locality in the physical generation of the application database, e.g., by clustering related data in the same memory segment.

The OO7 benchmark demonstrated the sensitivity of response time to database size and interconnectivity in the form of fanout connections between atomic parts. The other four structural features were held constant in the benchmark. (Use of locality was constant for a given platform, but varied from platform to platform in the Wisconsin study.)

In this study sensitivity to the degree of aggregation and inheritance was measured, while the other four features were held constant.

## SIMULATION APPROACHES TO MODELING OODB

Because the current study extended the OO7 benchmark with the addition of stochastic features, the literature was also reviewed for any simulation approaches to modeling OODB. [Brumfield et al, 1988] described the design of a simulation model for evaluating the performance of a distributed object-oriented database system. The objective of the model was to project the future performance for ORION-2, a distributed extension of the workstation-based OODB ORION-1 developed by the Advanced Computer Architecture Program at Microelectronics and Computer Technology Corporation (MCC). Two possible hardware configurations of the shared database were modeled: 1) a server model, with user nodes accessing only one server machine for shared data, and 2) a fully distributed model, in which shared data were distributed among all workstations. Query processing time and object access time were measured, while experimentally varying the following general workload features for each network configuration: 1) the percentage of global vs. local queries, and 2) the total number of nodes (processors) on the network.

Parameter values used in the simulation came partly from measurement of all applications running on ORION-1 at that time, and partly from estimates. Because all types of user applications were lumped together into a composite workload for this study, CPU time and network time were modeled as constants, depending on message size (small/medium/large). Other workload assumptions were made as to the probability distributions for number of objects processed per query, the location of objects (in memory, on local disk, on remote disk).

Characterization of the workload in this simulation study was at a higher level than in the current research study, because the objective was to identify system bottlenecks as a

function of network configuration, percentage of global queries, and number of nodes. The significance of this article to this research study was the identification of possible variables to be included in workload studies.

## ANALYZING PROCESSES TO IDENTIFY SOURCES OF VARIABILITY

A more general, systematic methodology for analyzing systems was found in [Melton, 1991]. In the field of quality control, process studies are defined as collecting data over time about any sort of system (in a broad sense) that transforms inputs into outputs, such as a manufacturing or service process utilizing people, materials, methods, equipment, etc. Variability in the inputs and outputs of a process is expected, and can be measured and analyzed in the process study. In particular, variability can be categorized as **common cause** (i.e., inherent in the process, from a multitude of sources, affects all outcomes) versus **special cause** (i.e., from an assignable source, affecting only one or a few outcomes from a process). The key steps to perform a process study are as follows:

### 1) Define the process to be studied:

Define the boundaries of the process being studied (the beginning and ending steps of the process). Define the current operation of the process (the flow of steps/events comprising the process). Define the important subprocesses to be studied.

### 2) Identify the attributes about the process to be studied:

These include first of all those outcome characteristics of importance to the user of the process. Then, characteristics of the inputs which may cause variability in the outcome are identified. Major categories of sources of variability may be materials,

machines, methods, people, environment, and measurements. Determine which input attributes hypothetically have the greatest impact on the outcome. Define the attributes as measurable variables.

### 3) Plan the data collection process:

Define the sampling plan and data collection methods. Pilot, then implement the data collection plan.

### 4) Analyze the data collected:

Determine whether the inputs and outputs are stable over time (is there evidence of special cause variability, or stable, common cause variability in run charts or control charts). Looking at outliers in the data as a function of time often aids in interpretation, moreso than looking at overall data distributions. Upward/downward trends and cyclical behavior can be detected. If there is stability, then additional statistical analyses are appropriate. Descriptive statistics (Pareto analysis, scatter diagrams) and multivariate statistics (covariance analysis, factor analysis) may be used.

The focus on major sources of variability, and use of some specialized analytical techniques (run charts and control charts, for example), add to traditional statistical techniques for computer system performance measurements and evaluation.

## SOFTWARE METRICS FOR OBJECT-ORIENTED APPLICATIONS

Because of limited results in the OODB performance literature characterizing the distinguishing features of OODB applications, the literature on software metrics was



examined for some means of evaluating and comparing object-oriented software designs. Research on software metrics for object-oriented applications is in the early stages, as traditional metrics are thought to be inappropriate.

[Chidamber and Kemerer, 1994] proposed six software metrics for object-oriented software design. The metrics were designed to measure the complexity in the design of classes for an application, and were intended to be implementation-independent. Although it was acknowledged that dynamic behavior (and thus performance) of a system may not be captured by these measures, the authors' objective was to provide metrics for use early in the life-cycle of an application, before program development. The metrics were designed to be predictors of the time and effort to develop, test, and maintain the classes of an application. The authors provided supporting empirical data collected from commercial development projects in two development environments.

The six metrics proposed by Chidamber and Kemerer were:

- **Weighted Methods Per Class (WMC)**--Each method in a class was assigned a complexity measure. WMC was the summation of complexity measures for all methods of a class.
- **Depth of Inheritance Tree (DIT)**--DIT was the maximum length from a class to its root class in an inheritance hierarchy. The deeper a class was in a hierarchy, the more methods would be inherited, making it more complex to predict its behavior.
- **Number of Children (NOC)**--NOC was the number of immediate sub-classes per class. The greater the number of children, the greater reusability through inheritance, and the greater requirements during testing.
- **Coupling Between Objects (CBO)**--The CBO for a class was the count of the number of couples with other classes, i.e., any time a method declared in one class used a method or variable defined by the other class. As with traditional system design,

excessive coupling between object classes was considered to be detrimental, preventing reuse and requiring more extensive testing.

- **Response for a Class (RFC)**--The response set of a class was a set of methods that could potentially be executed in response to a message received by an object of that class, defined only up to the first level of nesting of method calls. The larger the response set the greater the complexity of the class, requiring more testing and debugging.
- **Lack of Cohesion in Methods (LCOM)**--The degree of similarity of the methods within a class was defined as the intersection of the sets of instance variables used by the methods. The larger the number of similar methods, the more cohesive the class. Cohesiveness would promote encapsulation; lack of cohesion would increase complexity and indicate that classes should perhaps be split into multiple sub-classes.

Empirically, Chidamber and Kemerer showed that most classes have zero or few sub-classes. The explanation given again suggested that the designers were not using inheritance of methods as a basis for designing classes, and in fact "some C++ designers systematically avoid sub-classing in order to maximize operational performance." (This was the primary mention of performance in this paper otherwise oriented toward application development time, testing and maintenance.)

These metrics have provided a starting point for OODB performance research, and the list will be added to or subtracted from with subsequent research. For the measurement of system performance to do database operations (e.g., lookup, insertion, traversal), the complexity of methods for a class may not be relevant, since these methods may perform non-database work. For this study, however, the DIT and NOC metrics may be useful for classifying applications on the basis of degree of inheritance.

Aggregation was not dealt with explicitly with the six Chidamber and Kemerer metrics. Because aggregation is an important characteristic of many OODB applications, this type of specific metric was added in the current study.

## CHAPTER III

### RESEARCH METHOD

#### OBJECTIVES OF STUDY

Structural design features of an object-oriented database application were varied to explore the relationship between these features and performance (system response time). Sensitivity to the degree of aggregation and the degree of inheritance in the application were measured. Also, the statistical distribution of response times for database operations in a mixed transaction workload was measured.

#### OPERATING PLATFORM

The following platform was used for this study (all brand or product names are trademarks or registered trademarks of their respective holders):

##### Hardware, Operating System, and Compiler:

- Win Pentium 90 MHz Processor
- 16 Megabytes RAM
- 344 Megabyte Hard Disk Partition
- OS/2 Warp Operating System, Version 3
- IBM C/C++ compiler, version 2

##### Object-Oriented Database Management System

- ObjectStore Release 3.1 for OS/2, by Object Design, Inc., Burlington, MA

## TEST INSTRUMENT

The OO7 Benchmark application database and selected database operations were used as the test instrument in this study (Carey et al, 1993; Brownsmith, 1995, see Appendices B-D). By varying parameters which determined the degree of aggregation and the degree of inheritance implemented in this database, the impact of application design characteristics on performance was investigated. An overview of this benchmark was provided in Chapter 2.

## OVERVIEW OF THE EXPERIMENTAL DESIGN

### Variables Held Constant for All Three Experiments

The following decision variables were held constant for this study:

RAM	16 megabytes Running OS/2, ObjectStore, and OO7 Benchmark processes only
Hard Disk Space	344 megabyte OS/2 partition 150 - 190 megabytes free hard disk space for simulations after generation of each database
ObjectStore Client Cache	12,288,000 bytes
Fetch Policy	2 pages (8,192 bytes) for all database operations
Replacement Policy	least recently used
ObjectStore Server	presizing of collections off, os_toggle_mapaside enabled, opt_cache_lock_mode enabled, default settings for other server parameters

### Overview of OO7 Operations Selected for This Study

The following database operations were selected from the OO7 Benchmark for this study. These operations were chosen based on the information provided, the degree of realism when used with ObjectStore, and the ability to maintain database size and state during the experiment (i.e., perform no updates).

Table 3.1			
OVERVIEW OF OO7 OPERATIONS SELECTED FOR THIS STUDY			
<u>DATABASE OPERATION</u>	<u>USED TO TEST AGGREGATION</u>	<u>USED TO TEST INHERITANCE</u>	<u>USED TO TEST TXN WORKLOAD</u>
T1	YES	YES	YES
T6	YES	YES	YES
Q1			YES
Q2			YES
Q3			YES
Q4	YES		YES
Q5	YES		YES
Q6	YES	YES	YES
Q8			YES

## EXPERIMENTAL DESIGN FOR RESEARCH QUESTION 1: AGGREGATION

### Review of Research Question on Aggregation

What is the relationship between the degree of aggregation in an OODB user application and the response time to do database operations?

### Definition of Terms

#### 1. Aggregation

Aggregation is a form of association between objects in which an aggregate object is made of component objects. Component objects are thus “part of” the aggregate object. [Rumbaugh et al, 1991] This form of association is common in object-oriented database applications.

Aggregation is implemented in ObjectStore as a 1:many or many:many “part-of” relationship between two objects. ObjectStore treats the relationship itself as a class, so for any pair of related parts, there is a relationship object. This relationship object is of a constant size (4 bytes per member part), regardless of the size and type of the member parts. When ObjectStore processes the relationship, it processes the relationship objects. It does not access the actual member objects unless it needs to retrieve or change the value of an attribute of that object.

## 2. Degree of Aggregation

This is the number of “part-of” relationship objects that must be processed by a given database operation. Degree of aggregation can also be thought of as the “number of hookups” between part objects in OO7 that must be processed by a database operation.

## 3. Database Operations

The OO7 operations T1, T6, Q4, Q5, and Q6 processed objects in the OO7 database which were members of an aggregation relationship. These operations were used to test the hypothesis on aggregation.

### Research Hypothesis for Aggregation

H1: For database operations processing aggregated objects, response time is a function of the number of relationship objects processed.

### Operational Definitions: Dependent Variable

Response Time is defined as the seconds to perform a database operation like a query or traversal, excluding transaction overhead, under warm cache conditions.



### Operational Definitions: Independent Variable

# Hookups Processed is defined as the number of relationship objects processed by a database operation. Varies for each database operation, depending on which parts of the database are used.

### Manipulation of the Independent Variable

In the OO7 database generation program, there were several constants which could be set by the user to control the numbers of objects and hookups in the database.

#### 1. Number of Assembly Levels (e.g., 4)

Affected the depth of the aggregation tree, which affected any database operation which traversed some or all of the assemblies in the database.

#### 2. Number of Atomic Parts Per Composite Part (e.g., 100)

Increased the complexity of relationships within each Composite Part, since the number of Connections and the random interconnectivity between Atomic Parts increased with this parameter.

#### 3. Number of Assemblies Per Assembly (e.g., 3)

Affected the width/span of the aggregation tree, which affected any database operation which traversed some of all of the assemblies in the database.

#### 4. Number of Composite Parts per Base Assembly (e.g., 3)

Affected the number of hookups between composite parts and base assemblies.

Thus, a database generated using the parameters 4/100/3/3 has

Number of Assembly Levels	4
Number of Atomic Parts Per Composite Part	100
Number of Assemblies Per Assembly	3
Number of Composite Parts per Base Assembly	3

Nine databases were generated, manipulating the above parameters, with database size ranging from approximately 27 megabytes to 42 megabytes. Table 3.2 summarizes the setting of the constants in the generation of the databases, and the resulting levels for the independent variable # Hookups Processed. The constants were chosen 1) to generate databases larger than the available amount of RAM, 2) to approximate the range of sizes of the OO7 database used in other research studies found in the literature review, and 3) to give an adequate number of levels of the independent variable to develop a regression model, per operation, expressing the relationship between response time and the number of hookups processed by that operation.

Three of the nine databases were chosen for validating the regression models. One of these databases, 5/150/3/3, provided levels of the independent variable in the middle of the range of data used to develop the models. Another database used for validation, 7/150/3/3, was chosen to see how well response time could be predicted at the highest

values of the independent variable, and beyond the range of data used to develop the regression models. Lastly, the database 6/100/4/2 was chosen to see if the # Hookups Processed could be generalized to predict response time for databases with a different aggregation tree structure (wider and more shallow).

Table 3.2  
LEVELS CHOSEN FOR INDEPENDENT VARIABLE,  
# HOOKUPS PROCESSED

Database <u>Generation Constants</u>	# Hookups Processed				
	<u>T1</u>	<u>T6</u>	<u>Q4</u>	<u>Q5</u>	<u>Q6</u>
For Modelling:					
4/100/3/3	8,221	202	81	81	121
4/150/3/3	12,271	202	81	81	121
5/100/3/3	24,664	607	243	243	364
6/100/3/3	73,993	1,822	729	729	1,093
6/150/3/3	110,443	1,822	729	729	1,093
7/100/3/3	221,980	5,467	2,187	2,187	3,280
For Validation of Model:					
5/150/3/3	36,814	607	243	243	364
7/150/3/3	331,330	5,467	2,187	2,187	3,280
6/100/4/2	208,213	5,461	2,048	2,048	3,413

### Methodology for Data Collection

For each of the nine experimental databases, the following steps were repeated:

1. Generate the database using the desired constants
2. For each of the five operations of interest,

Begin the database transaction,  
For 30 repetitions of the operation  
{

```

    Measure the start time;
    Do the database operation;
    Measure the stop time;
    }
Commit the database transaction.

```

Thus, the response time was measured for each individual repetition of a database operation, excluding transaction overhead. The first (and in some cases, second) repetition of the operation was needed to warm the cache; subsequent repetitions were under warm conditions.

#### Data Analysis: Model and Statistical Hypothesis

The data from the six databases to be used for the regression models were combined into one dataset. The first observation (and in some cases, the second) for each operation in each database was deleted, to keep only steady state observations. The dataset was then sorted by database operation.

For each database operation, the following regression model was developed:

$$Y_i = B_0 + B_1 X_{i1} + E_i$$

The statistical hypothesis to be tested was:

$$H_0: B_1 = 0$$

$$H_a: B_1 \neq 0$$

## EXPERIMENTAL DESIGN FOR RESEARCH QUESTION 2: INHERITANCE

### Review of Research Question on Inheritance

What effect does the degree of inheritance in an OODB user application have on response time?

### Definition of Terms

#### 1. Inheritance

There were two ways inheritance was carried out in the OO7 Benchmark. First, there was inheritance of data attributes from superclasses to subclasses. This affected the size of the objects stored in the database. In ObjectStore, if an object inherited data attributes from a superclass, its size was greater than it would be if the same attributes were declared directly in the subclass, without inheritance.

In OO7, there was also inheritance of methods through run-time polymorphism. For example, which traversal method was executed during a T1 or T6 operation depended on the type of object being traversed. This decision was made at main memory speeds at run-time.

#### 2. Degree of Inheritance

Existing metrics from [Chidamber and Kemerer, 1994] measured the degree of inheritance in an application's class definitions. Depth in Tree (DIT) was 1 for the OO7

application in its original implementation. (DIT measured the maximum length from a class to its root class in the overall class hierarchy for the application.)

Alternatively, the Number of Children metric from [Chidamber and Kemerer, 1994] measured the number of classes which were subclasses of another class. NOC was 2 in the original OO7 implementation.

A new metric was added in this study to measure the degree of inheritance in the execution of an application's methods. This metric measured the use of virtual functions which were resolved at run-time either by using a pointer to an object directly (no inheritance) or by using a pointer to its superclass (inheritance). This metric, called Depth of Run-Time Resolution for the purposes of this study, was the maximum number of levels in an inheritance hierarchy which must be resolved at run-time when executing virtual functions.

### 3. Database Operations

The OO7 database operations T1, T6, and Q6 processed objects which were members of an inheritance hierarchy, with inherited data attributes and/or virtual functions. These three operations were used to test the hypotheses on the effect of inheritance on response time.

#### Research Hypotheses for Inheritance

H2: When class definitions have increased inheritance of data attributes among objects, response time to perform database operations on such objects will increase.

H3: For database operations using virtual functions, there is no significant difference in response time as the Depth of Run-Time Resolution increases.

#### Operational Definitions: Dependent Variable

Response Time is defined as the seconds to perform a database operation like a query or traversal, excluding transaction overhead, under warm cache conditions.

#### Operational Definitions: Independent Variables

Number of Children is defined as the number of classes which are subclasses of another class.

Depth of Run-Time Resolution is defined as the maximum number of inheritance level resolved through run-time polymorphism in an application's methods.

#### Manipulation of the Independent Variables

In the original OO7 C++ programs, the class definitions implemented  $NOC = 2$  (the abstract class Assembly had two subclasses, Base Assembly and Complex Assembly). The virtual function for traversals implemented  $Depth\ of\ Run-Time\ Resolution = 1$  (a pointer to an Assembly was resolved at run-time to be either a pointer to a Base or Complex Assembly). See Figure 2.1.

But for a single user application, with given logical semantics, designers have a choice in how much to use inheritance in the physical implementation (i.e., when coding the class definitions and the function definitions in C++).

For example, in the original OO7 Benchmark Database application, the Module, Complex Assemblies, Base Assemblies, and Composite Parts had several common data members (id, type, buildDate) and one common member function (traverse). (Atomic Part had similar members, but with slightly different declarations.) However, in the original C++ implementation of the OO7 Benchmark, only Complex Assemblies and Base Assemblies inherited the id, type, buildDate data members and the traverse pure virtual function from an abstract parent class "Assembly."

In this study, the class definitions were revised to add superclasses DesignObject and RootObject. This increased NOC to 10, so all object classes would inherit at least from the top superclass in the hierarchy, RootObject. The new class definitions also added the potential to increase Depth of Run-Time Resolution to 3. Figures 3.1 and 3.2 demonstrate the revision of the class hierarchy to add inheritance, and Appendix C contains the revised C++ class definitions.

Then, the traversal program was modified to implement run-time polymorphism with a pointer to RootObject, to be resolved at run time depending on whether the object to be traversed was Module, Complex Assembly, Base Assembly, Composite Part, or Atomic Part. Using this revised traversal program, the Depth of Run-Time Resolution would be 3. The modified traverse program is in Appendix D.



Figure 3.1  
 OO7 OBJECT MODEL  
 Modified To Add a Second Level of Inheritance (DesignObject)

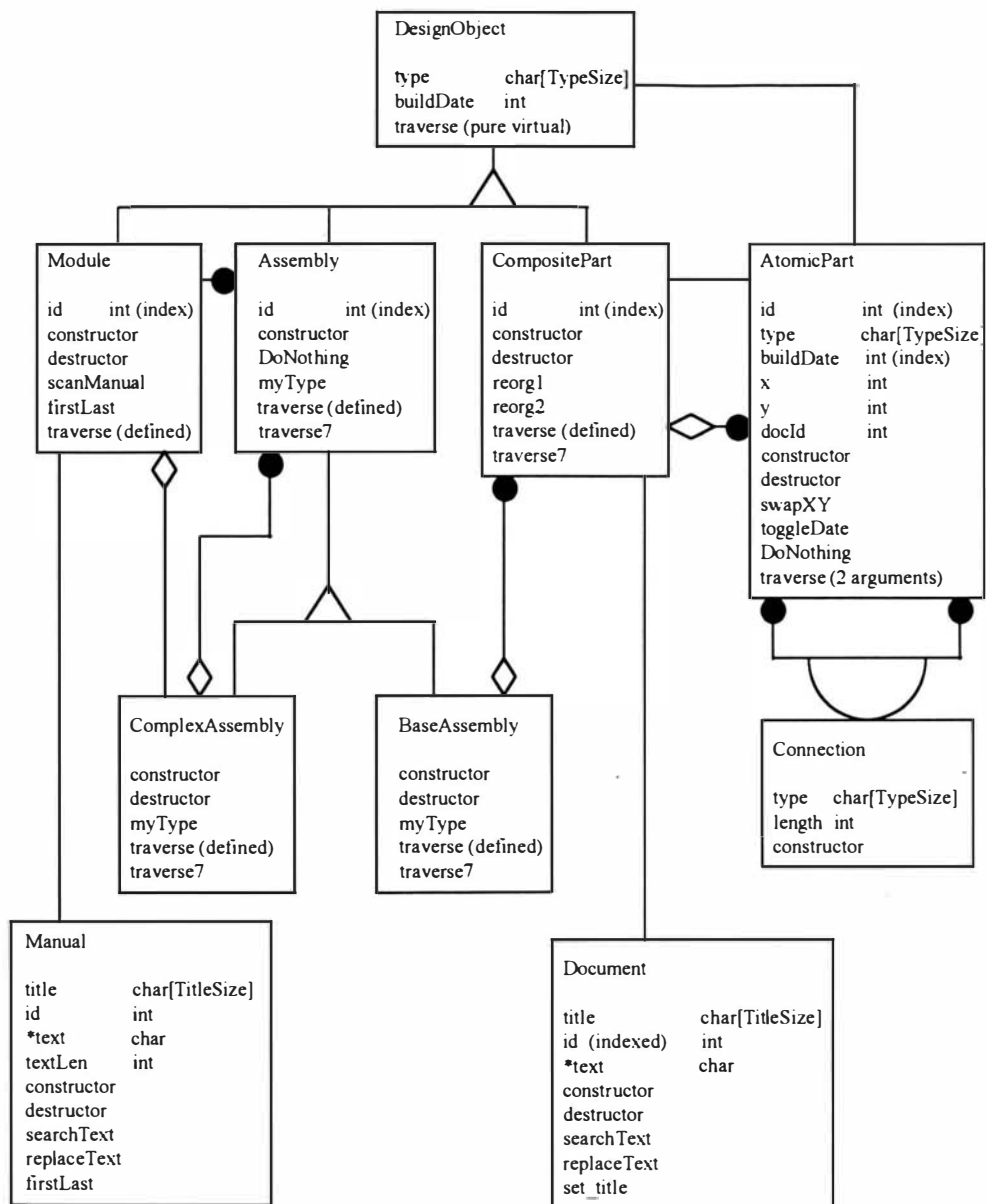
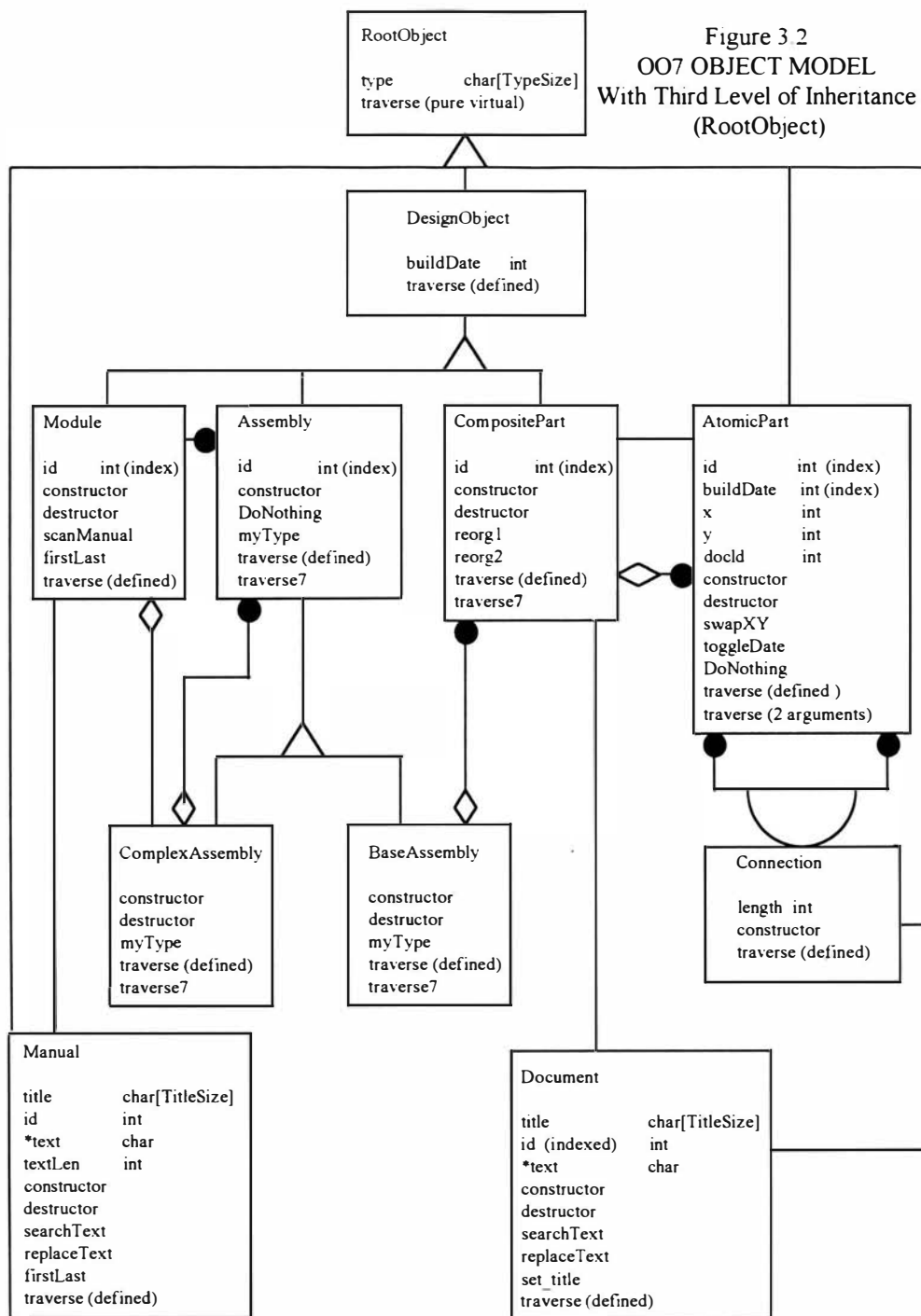


Figure 3 2  
OO7 OBJECT MODEL  
With Third Level of Inheritance  
(RootObject)



The following three experimental treatments were then simulated:

Table 3.3	
MANIPULATION OF THE INDEPENDENT VARIABLES TO TEST THE EFFECTS OF INHERITANCE	
TREATMENT 1	TREATMENT 2
NOC = 2	NOC = 10
Depth of Run-Time Resolution = 1	Depth of Run-Time Resolution = 1
Original OO7 class definitions	New OO7 class definitions
Original OO7 traversal program	Original OO7 traversal program
(Original OO7 class definitions and new traversal program not syntactically possible)	TREATMENT 3
	NOC = 10
	Depth of Run-Time Resolution = 3
	New OO7 class definitions
	New traversal program

### Methodology for Data Collection

For each of the three experimental treatments, the following steps were performed:

Treatment 1:

1. Generate the database 5/150/3/3 using the original OO7 class definitions.

2. Using the original OO7 traversal program, for each of the three operations of interest,

```

    Begin the database transaction;
    For 30 repetitions of the operation
    {
        Measure the start time;
        Do the database operation;
        Measure the stop time;
    }
    Commit the database transaction.

```

As in the aggregation experiment, the response time was measured for each individual repetition of a database operation, excluding transaction overhead. The first (and in some cases, second) repetition of the operation was needed to warm the cache; subsequent repetitions were under warm conditions.

#### Treatment 2:

1. Modify the class definitions to implement  $NOC = 10$ .
2. Compile the application with the new class definitions and the original traversal program.
3. Generate the database 5/150/3/3 using the new OO7 class definitions.

4. Using the new traversal program, for each of the three operations of interest:

```

Begin the database transaction;
For 30 repetitions of the operation
{
    Measure the start time;
    Do the database operation;
    Measure the stop time;
}
Commit the database transaction.

```

### Treatment 3:

1. Use the class definitions to implement  $NOC = 10$  from Treatment 2.
2. Compile the application using the new class definitions and the new traversal program resolving all object traversals from a RootObject pointer.
3. Generate the database 5/150/3/3 using the new OO7 class definitions.
4. For each of the three operations of interest:

```

Begin the database transaction;
For 30 repetitions of the operation
{
    Measure the start time;
    Do the database operation;
    Measure the stop time;
}
Commit the database transaction.

```

### Data Analysis: Model and Statistical Hypotheses

The data from the three database experiments were combined into one dataset. The first observation (and in some cases, the second) for each operation in each database was deleted, to keep only steady state observations.

For each database operation, the following one-way ANOVA model was developed:

$$Y_{ij} = \mu_i + E_{ij}$$

The statistical hypotheses to be tested were:

Increasing NOC (Treatment 1 vs. Treatment 2)

$$H_0: \mu_1 = \mu_2$$

$$H_a: \mu_1 \neq \mu_2$$

Increasing Depth of Run-Time Resolution (Treatment 2 vs. Treatment 3)

$$H_0: \mu_2 = \mu_3$$

$$H_a: \mu_2 \neq \mu_3$$

## EXPERIMENTAL DESIGN FOR RESEARCH QUESTION 3: WORKLOAD SEQUENCE

### Review of Research Question on Workload Sequence

How does the sequence of database operations in a user transaction workload influence response time variability?

### Definition of Terms

#### 1. User Transaction Workload

A user transaction workload is defined as a randomly-generated sequence of database operations, each operation within its own transaction boundaries (one operation per transaction, unlike the experiments on aggregation and inheritance). In this study, a workload was simulated in which each of nine selected database operations had an approximately equal chance of being executed next in the sequence. The nine queries and traversals chosen for the workload mix processed different segments of the database, but did not modify any of the data. Work per operation was thus repeatable.

#### 2. Database Operations

The following OO7 database operations were selected to test the hypotheses on workload sequence, based on their realism, their variety, and the ability to maintain the state of the database: T1, T6, Q1, Q2, Q3, Q4, Q5, Q6, Q8

### 3. Response Time Variability

Response time variability in this stochastic process was influenced by the state of the client cache and main memory. Because each operation was executed under a variety of memory states, this study simulated the range of possible response times a single user would experience for each query or traversal.

#### Research Hypotheses for Workload Sequence

##### 1. Effects of Increasing the Size of the Working Set for Each Database Operation

H4: Response time is a function of the number of workload units processed by a given database operation.

##### 2. Effects of Preceding Database Operations

H5: An operation will have a significantly higher response time when the immediately preceding operation flushes the client cache with a completely disparate working set.

H6: An operation will have a significantly higher response time when it is the second operation following one which flushes the client cache with a completely disparate working set.

H7: An operation will have a significantly higher response time when the immediately preceding operation flushes the client cache, even with a partly usable working set.



H8: An operation will have a significantly higher response time when it is the second operation following one which flushes the client cache, even with a partly usable working set.

H9: An operation will have a significantly lower response time when preceded by itself.

### 3. Significance of Overall Model

H10: There is a significant relationship between response time and the set of independent variables used in the workload sequence model.

#### Operational Definitions: Dependent Variable

Response Time is defined as the seconds to perform a database operation like a query or traversal, including transaction overhead, and under any cache conditions which may occur in a mixed transaction workload. This response time reflects the response time a user would observe under realistic working conditions.

#### Operational Definitions: Independent Variables

For H4:

The independent variable is the Number of Workload Units processed by a given database operation. The definition of this variable depends on the database operation; for

some operations the number of workload units equals the number of relationship objects processed, for others it equals the number of parts processed.

T1	Number of relationship objects (# Hookups)
T6	Number of relationship objects (# Hookups)
Q4	Number of relationship objects (# Hookups)
Q5	Number of relationship objects (# Hookups)
Q6	Number of relationship objects (# Hookups)
Q1	Number of atomic parts in index
Q2	Number of atomic parts in index
Q3	Number of atomic parts in index
Q8	Number of atomic parts (does not use index)

Nine databases were generated to vary the levels of this independent variable; seven of the nine were generated to collect data for the models, and the other two were generated to collect data for validating the models. Table 3.4 illustrates the levels studied for the independent variable, Number of Workload Units.

For H5 - H9:

The independent variables are defined using dummy variables which indicate the presence of preceding operations which may heavily influence response time. These variables are not manipulated; they occur randomly in the workload.

Table 3.4  
LEVELS CHOSEN FOR INDEPENDENT VARIABLE,  
# WORKLOAD UNITS = # HOOKUPS PROCESSED

Database <u>Generation Constants</u>	# Hookups Processed				
	<u>T1</u>	<u>T6</u>	<u>Q4</u>	<u>Q5</u>	<u>Q6</u>
For Modelling:					
4/100/3/3	8,221	202	81	81	121
4/150/3/3	12,271	202	81	81	121
4/200/3/3	16,321	202	81	81	121
5/100/3/3	24,664	607	243	243	364
5/200/3/3	48,964	607	243	243	364
7/100/3/3	221,980	5,467	2,187	2,187	3,280
7/150/3/3	331,330	5,467	2,187	2,187	3,280
For Validation of Model:					
5/150/3/3	36,814	607	243	243	364
7/200/3/3	440,680	5,467	2,187	2,187	3,280

LEVELS CHOSEN FOR INDEPENDENT VARIABLE,  
# WORKLOAD UNITS = # ATOMIC PARTS PROCESSED

Database <u>Generation Constants</u>	# Atomic Parts Processed
	<u>Q1, Q2, Q3, Q8</u>
For Modelling:	
4/100/3/3	50,000
4/150/3/3	75,000
4/200/3/3	100,000
5/100/3/3	50,000
5/200/3/3	100,000
7/100/3/3	50,000
7/150/3/3	75,000
For Validation of Model:	
5/150/3/3	75,000
7/200/3/3	100,000

Follows Q8 (Goes with H5)

For each operation in the workload that immediately follows a Q8, this dummy variable has a value of 1. Otherwise the value is 0. (Q8 flushes the client cache with a completely disparate working set.)

Two After Q8 (Goes with H6)

For each operation in the workload that is the second operation after a Q8, this dummy variable has a value of 1. Otherwise the value is 0.

Follows T1

For each operation in the workload that immediately follows a T1, this dummy variable has a value of 1. Otherwise the value is 0. (T1 flushes the client cache with a potentially usable working set.)

Two After T1

For each operation in the workload that is the second operation after a T1, this dummy variable has a value of 1. Otherwise the value is 0.

## Follows Itself

For each operation in the workload that immediately follows itself, this dummy variable has a value of 1. Otherwise the value is 0.

## Methodology for Data Collection

### 1. Generate the Randomized Transaction Workload

A sequence of 500 database operations was generated in a pseudorandom order from the distribution in Table 3.5, and saved to a file. Each operation had approximately an equal chance of occurring in this workload. The program for generating the sequence of database operations is in Appendix A. This program used the implementation of Lehmer's multiplicative linear congruential random number generator found in (Parks and Miller, 1985), ported to the current experimental platform in C.

This identical sequence of database operations was used for all experiments, so that the same workload was presented to each database configuration. These observations allowed an analysis of the variation of response times for each type of operation. Running each database operation in this order randomized the cache effects from one operation to the next, and simulated the average and range of response times which would be experienced in a single-user workload.

Table 3.5

## DISTRIBUTION OF DATABASE OPERATIONS

<u>Operation</u>	<u>Probability</u>	<u>Expected Frequency</u>	<u>Observed Frequency</u>
T1	.11	55	53
T6	.11	55	56
Q1	.12	60	56
Q2	.11	55	52
Q3	.11	55	69
Q4	.11	55	46
Q5	.11	55	75
Q6	.11	55	45
Q8	<u>.11</u>	<u>55</u>	<u>48</u>
	1.00	500	500

T1 was the first operation run, given the above probability distribution and a seed of 1. This operation essentially flushed the client cache, since its working set was greater than 12.8 MB. This provided a known, repeatable starting state for each experiment. T1 also warmed the cache for any subsequent operation that used primarily atomic parts, so a steady state was reached quickly.

After generating the sequence of random operations, the testing for each database proceeded as follows:

2. Delete any previously tested database files and results files, to keep free hard disk space approximately constant (affects seek time and OS/2 swap file size).
3. Set the following environmental variables:  
 set dbname=filename (where filename is the database file)  
 set os\_toggle\_mapaside=1 (to optimize ObjectStore performance under OS/2)

4. Generate the database, using the appropriate file of configuration parameters.
5. Record object counts for the database, time to generate the database, and database size.
6. Run ObjectStore's ossize utility and save to a file. This contains structural information about the size and contents of the hard disk segments allocated for the database.
7. Run the simulation program (Appendix B), using the random sequence of 500 operations as input. Each database operation is run by itself within a database transaction, and timed. Save the operation type, the transaction time, and the count of objects processed by each operation to a results file.

#### Data Analysis: Model and Statistical Hypotheses

For each database operation, the following multiple regression model was developed by combining the results from the seven databases described in Table 3.4. The first 15 observations from each database's simulation were deleted, since they were initialization conditions before reaching a steady state. The results from the other two databases, 5/150/3/3 and 7/200/3/3, were held out for validating the regression model.

$$Y_i = B_0 + B_1X_{i1} + B_2X_{i2} + B_3X_{i3} + B_4X_{i4} + B_5X_{i5} + B_6X_{i6} + E_i$$

where

$Y_i$  is the response time  $Y$  in the  $i$ th trial,  
 $B_0$  is the  $Y$  intercept of the regression plane,

$X_{i1}$  is the value of the independent variable # Workload Units in the  $i$ th trial,  
 $X_{i2}$  is the value of the independent variable Follows Q8 in the  $i$ th trial,  
 $X_{i3}$  is the value of the independent variable Two After Q8 in the  $i$ th trial,  
 $X_{i4}$  is the value of the independent variable Follows T1 in the  $i$ th trial,  
 $X_{i5}$  is the value of the independent variable Two After T1 in the  $i$ th trial,  
 $X_{i6}$  is the value of the independent variable Follows Itself in the  $i$ th trial,  
 $B_1$  through  $B_6$  are the partial derivatives expressing the change in  $Y$  with the change in one independent variable holding the others constant, and  
 $E_i$  is the random error of the  $i$ th observation.

The statistical hypotheses to be tested for the six regression coefficients  $k = 1 \dots 6$  were:

$$H_0: B_k = 0$$

$$H_a: B_k \neq 0$$

The statistical hypothesis to be tested for the significance of the regression relation between the dependent variable and the set of  $X$  variables was:

$$H_0: B_1 = B_2 = \dots = B_6$$

$$H_a: \text{not all } B_k = 0$$

For this hypothesis, the  $F$  test was used.

For a given database operation, the distribution of response times was analyzed to determine the distribution shape. Also, for each operation, a linear graph of the time-ordered occurrences of the operation was analyzed, to understand how response times were affected by preceding events.



## CHAPTER IV

### RESULTS

#### RESULTS FOR RESEARCH QUESTION 1: AGGREGATION

Response time was linear with respect to the number of relationship objects processed (# Hookups) over the range of data tested, as Tables 4.1 - 4.7 illustrate. For Q4, Q5, Q6, and T6, the regression models were highly significant, as indicated by the F statistics. There was very little variation about the regression line for these database operations under warm cache conditions.

For Q4, Q5, Q6, and T6, there also was a very strong relationship between the # Hookups and response time, as indicated by the P-values for the  $B_1$  coefficients for the slope of the least-squares regression line. The regression equation was an effective predictor for the results of the runs held out for validation of the models.

Q4, Q5, Q6, and T6 were database operations which had a relatively small working set, compared to the size of the ObjectStore client disk cache (12.3 megabytes) and available RAM. However, in the case of T1, different results were obtained with the regression model (Table 4.5). When all databases were combined to build the model, one database (6/150/3/3) had response times for T1 radically different from the other databases. This particular database was different from smaller databases in that it was the

first database in which the size of the atomic parts segment processed during T1 exceeded the size of the client cache and available RAM. Consequently, significantly more paging activity was going on during T1 processing. Table 4.8 describes the physical segments in the OO7 database, including the size per segment and the contents and dynamic usage of each segment.

Investigating this further, if the data from all databases with only 100 atomic parts per composite part were plotted separately, the results were different. For this group, T1 response times were highly linear with # Hookups, with little variance around the line. This would be expected because with 100 atomic parts per composite, the database segment containing atomic parts is only about 10 megabytes.

Furthermore, if the data from the databases with 150 atomic parts per composite part were plotted separately, a different regression line was obtained. For the small databases, 4/150/3/3 and 5/150/3/3, the atomic parts segment was about 16 megabytes, but it was not fully traversed during T1 because not all of the 500 composite parts were hooked up to a base assembly. (See Table 4.9.) In databases 6/150/3/3 and 7/150/3/3, however, all composite parts were connected to at least one base assembly, so the recursive traversal over the entire tree structure of the database resulted in more paging activity to bring in the 16 megabyte database segment of atomic parts.

TABLE 4.1  
Q4 AGGREGATION MODEL

SUMMARY OUTPUT Q4

<i>Regression Statistics</i>	
Multiple R	1.00
R Square	1.00
Adjusted R Square	1.00
Standard Error	0.02
Observations	174

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	1	30.40	30.40	100781.63	4.84E-240
Residual	172	0.05	3.02E-04		
Total	173	30.45			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	0.01	1.79E-03	6.06	8.42E-09	7.33E-03	1.44E-02
#Hookups	5.74E-04	1.81E-06	317.46	4.84E-240	5.70E-04	5.77E-04

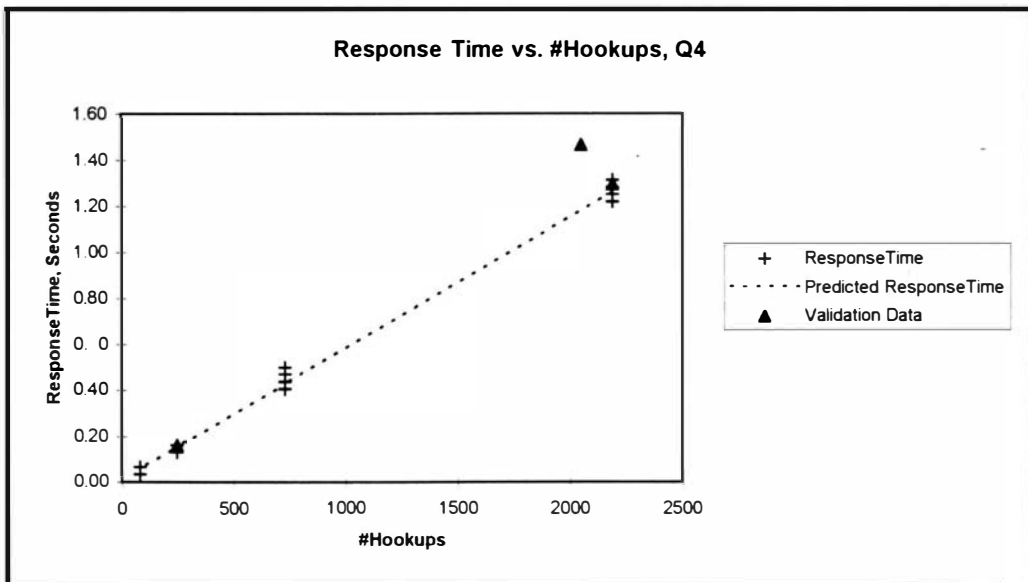


TABLE 4.2  
Q5 AGGREGATION MODEL

SUMMARY OUTPUT Q5

<i>Regression Statistics</i>	
Multiple R	0.98
R Square	0.96
Adjusted R Square	0.96
Standard Error	0.01
Observations	174

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	1	0.75	0.75	4698.30	8.26E-127
Residual	172	0.03	1.60E-04		
Total	173	0.78			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	3.21E-04	1.31E-03	0.25	0.81	-2.26E-03	2.90E-03
#Hookups	9.03E-05	1.32E-06	68.54	8.26E-127	8.77E-05	9.29E-05

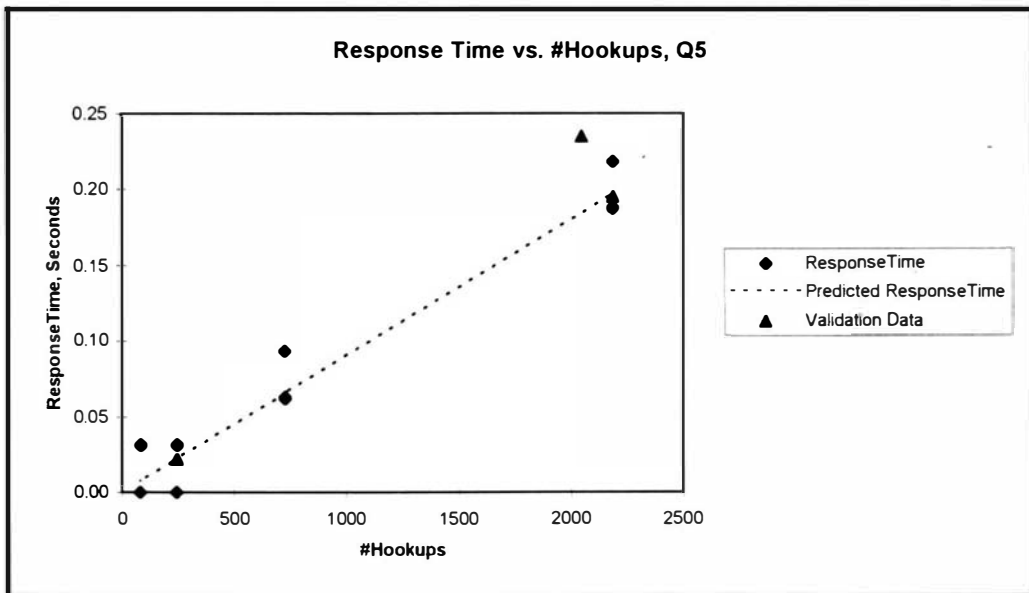


TABLE 4.3  
Q6 AGGREGATION MODEL

SUMMARY OUTPUT      Q6

<i>Regression Statistics</i>	
Multiple R	0.98
R Square	0.97
Adjusted R Square	0.97
Standard Error	0.01
Observations	174

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	1	0.65	0.65	4924.03	1.68E-128
Residual	172	0.02	1.31E-04		
Total	173	0.67			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	1.37E-04	1.18E-03	0.12	0.91	-2.20E-03	2.47E-03
#Hookups	5.58E-05	7.95E-07	70.17	1.68E-128	5.42E-05	5.73E-05

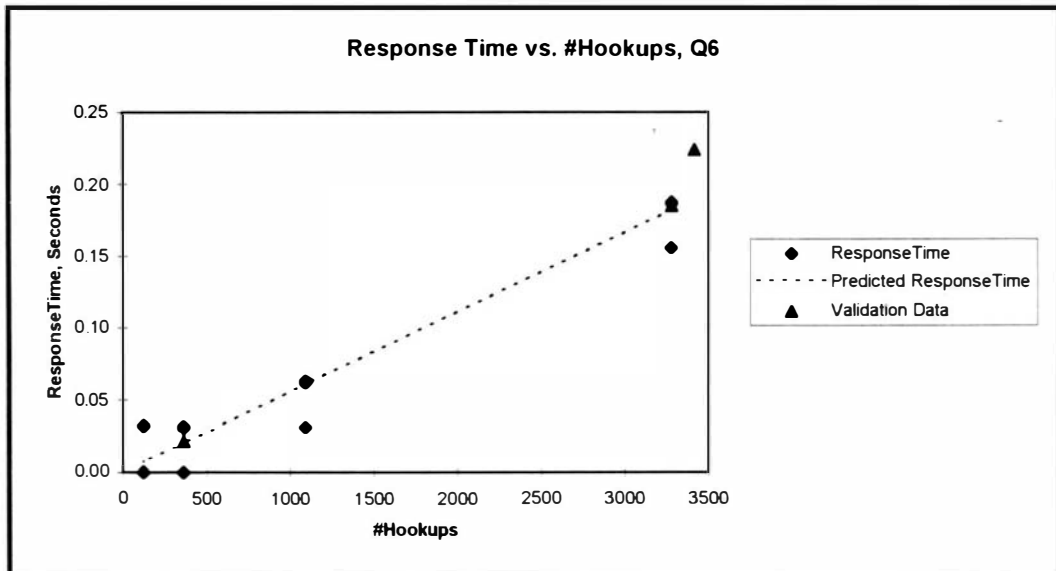


TABLE 4.4  
T6 AGGREGATION MODEL

SUMMARY OUTPUT T6

<i>Regression Statistics</i>	
Multiple R	1.00
R Square	0.99
Adjusted R Square	0.99
Standard Error	0.01
Observations	174

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	1	5.57	5.57	29888.90	8.55E-195
Residual	172	0.03	1.86E-04		
Total	173	5.60			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	-6.56E-04	1.41E-03	-0.47	0.64	-3.44E-03	2.13E-03
#Hookups	9.82E-05	5.68E-07	172.88	8.55E-195	9.71E-05	9.94E-05

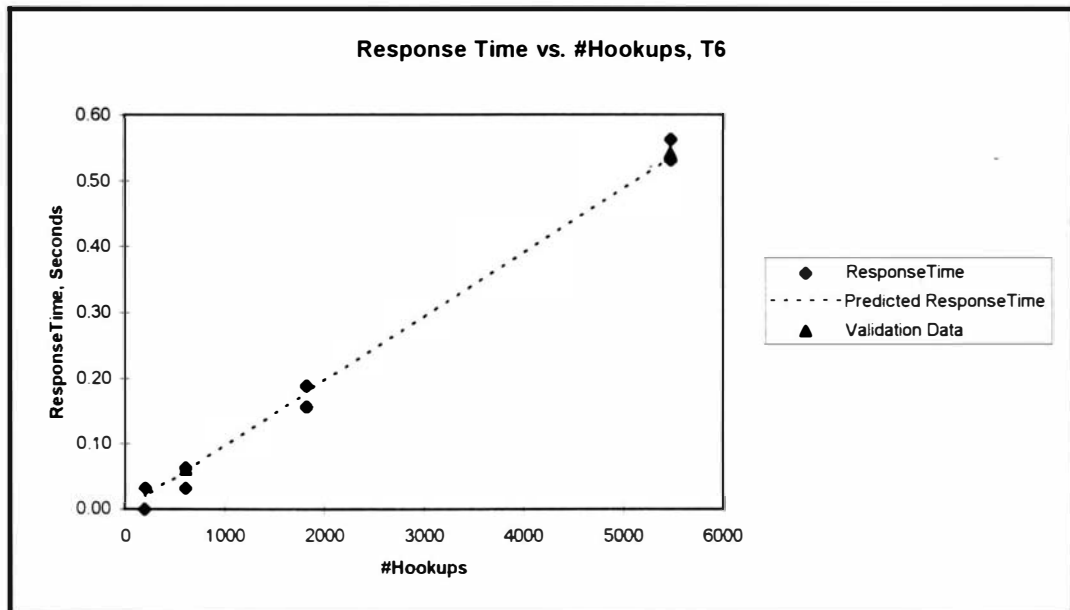


TABLE 4.5  
T1 AGGREGATION MODEL  
All Databases Combined

SUMMARY OUTPUT T1

<i>Regression Statistics</i>	
Multiple R	0.65
R Square	0.42
Adjusted R Square	0.42
Standard Error	21.56
Observations	171

ANOVA					
	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	1	58031.20	58031.20	124.89	4.61E-22
Residual	169	78524.37	464.64		
Total	170	136555.57			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	5.25	2.32	2.26	0.03	0.66	9.83
#Hookups	2.46E-04	2.20E-05	11.18	4.61E-22	2.03E-04	2.90E-04

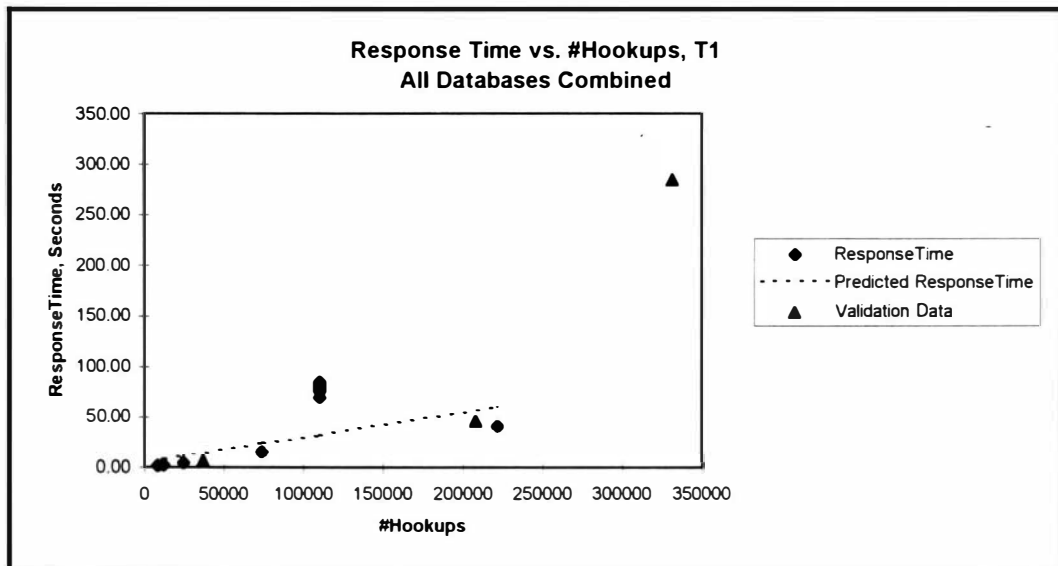


TABLE 4.6  
T1 AGGREGATION MODEL  
Databases with 100 Atomic Parts per Composite Part

SUMMARY OUTPUT      T1

<i>Regression Statistics</i>	
Multiple R	1.00
R Square	1.00
Adjusted R Square	1.00
Standard Error	0.43
Observations	114

ANOVA					
	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	1	26902.16	26902.16	146974.11	1.77E-176
Residual	112	20.50	0.18		
Total	113	26922.66			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	0.31	0.06	5.53	2.08E-07	0.20	0.42
#Hookups	1.83E-04	4.77E-07	383.37	1.77E-176	1.82E-04	1.84E-04

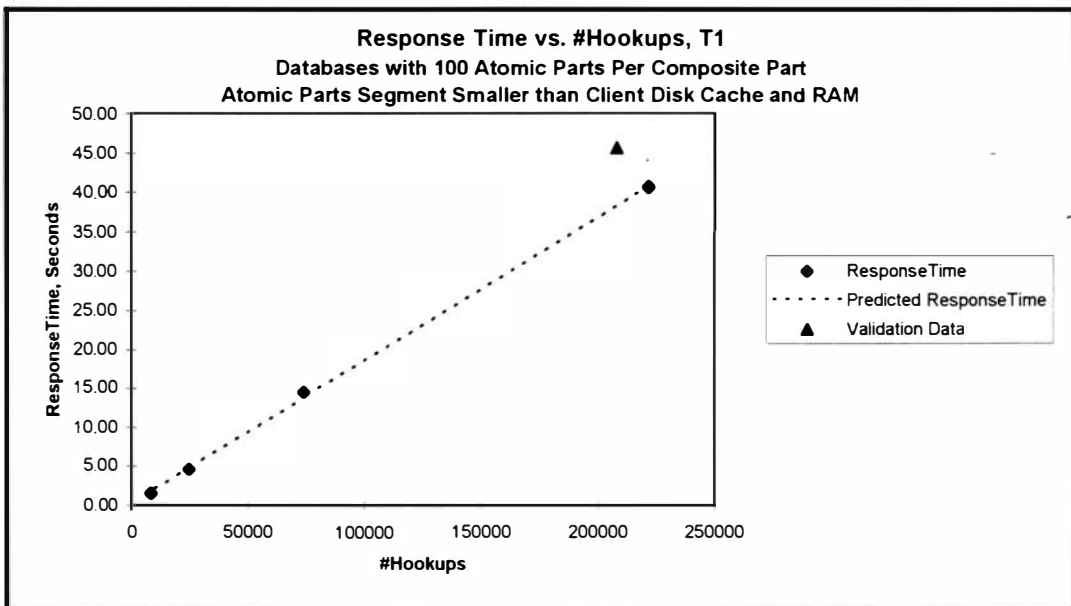




TABLE 4.7  
T1 AGGREGATION MODEL  
Databases with 150 Atomic Parts per Composite Part

SUMMARY OUTPUT T1

<i>Regression Statistics</i>	
Multiple R	1.00
R Square	1.00
Adjusted R Square	1.00
Standard Error	2.10
Observations	57

ANOVA					
	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	1	85375.77	85375.77	19419.66	8.51E-72
Residual	55	241.80	4.40		
Total	56	85617.57			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	-7.43	0.44	-16.84	2.27E-23	-8.31	-6.54
#Hookups	7.89E-04	5.66E-06	139.35	8.51E-72	7.77E-04	8.00E-04

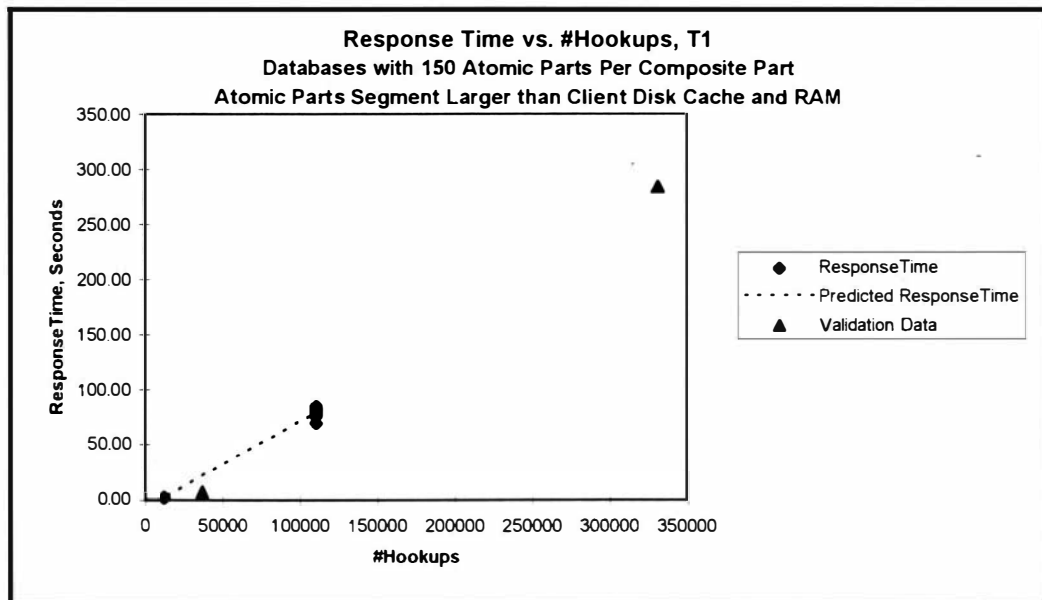


Table 4 8  
Analysis of ObjectStore Physical Segments  
Storing The OO7 Database  
(Databases 4/100/3/3 and 7/200/3/3)

<u>Segment Number</u>	<u>Approximate Segment Size (MB), 4/100/3/3</u>	<u>Approximate Segment Size (MB), 7/200/3/3</u>	<u>Objects Stored in Segment</u>	<u>Operations Accessing Segment</u>
0	0.1	0.1	Schema Segment	
2	0.1	0.4	BaseAssembly, ComplexAssembly Module	T1, T6, Q4, Q5, Q6
4	10.7	21.3	Connection, AtomicPart, CompositePart	<b>T1 (all),</b> T6 (part), Q1, Q2, Q3, Q5, Q6, Q8
6	12.9	12.9	Document, Manual	<b>Q8 (all),</b> Q4 (part)

Client Disk Cache Size = 12.2 MB, Total RAM Size = 16 MB

Table 4.9  
Comparison of Database Object Counts and Size

<u>Database Identifier</u>	<u>Size (MB)</u>	<u>#Complex Assemblies</u>	<u>#Base Assemblies</u>	<u>#Composite Parts</u>	<u>#Atomic Parts</u>
4/100/3/3	27.7	13	27	500	50,000
4/150/3/3	35.0	13	27	500	75,000
4/200/3/3	41.1	13	27	500	100,000
5/100/3/3	27.7	40	81	500	50,000
5/150/3/3	35.0	40	81	500	75,000
5/200/3/3	41.1	40	81	500	100,000
7/100/3/3	27.7	364	729	500	50,000
7/150/3/3	35.0	364	729	500	75,000
7/200/3/3	42.2	364	729	500	100,000

## RESULTS FOR RESEARCH QUESTION 2: INHERITANCE

### Treatment 1 vs. Treatment 2 (Effect of Class Definitions)

For Q6 and T6, with relatively small working sets, and relatively little repetitive processing of objects with inherited attributes or methods, there was no significant difference between Treatment 1 and Treatment 2. The response times were almost identical between the two groups. ANOVA results are summarized in Tables 4.10 - 4.12.

For T1, with extensive recursive and repetitive processing of objects with inherited attributes, there was a statistically significant difference between Treatment 1 (NOC = 2)

and Treatment 2 (NOC = 10). However, there was not an important difference in the response times (6.71 seconds vs. 6.73 seconds) for T1 on the 5/150/3/3 database. There was negligible within-group variation for operations under warm cache conditions, so a small between-group variation was statistically significant. It is possible that a statistically significant difference for T1 would result in a meaningful difference in response times with larger databases.

#### Treatment 2 vs. Treatment 3 (Effect of Run-Time Polymorphism)

Again for Q6 and T6, there was no significant difference between Treatment 2 (Depth of Run-Time Resolution = 1) and Treatment 3 (Depth of Run-Time Resolution = 3). The response times were almost identical between the two groups.

For T1, there was a strong statistically significant difference between Treatment 2 and Treatment 3. However, again there really was not an important difference in the response times (6.73 seconds vs. 6.90 seconds) for T1 on the 5/150/3/3 database. It is possible that a statistically significant difference for T1 would result in a meaningful difference in response times with larger databases.

TABLE 4.10  
ANOVA FOR Q6

TREATMENT 1 VS. TREATMENT 2

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Treatment1	29.0000	0.6250	0.0216	0.0002
Treatment2	29.0000	0.5930	0.0204	0.0002

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	0.0000	1.0000	0.0000	0.0794	0.7791	4.0130
Within Groups	0.0125	56.0000	0.0002			
Total	0.0125	57.0000				

TREATMENT 2 VS. TREATMENT 3

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Treatment2	29.0000	0.5930	0.0204	0.0002
Treatment3	29.0000	0.5940	0.0205	0.0002

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	0.0000	1.0000	0.0000	0.0001	0.9931	4.0130
Within Groups	0.0128	56.0000	0.0002			
Total	0.0128	57.0000				

TABLE 4.11  
ANOVA FOR T6

TREATMENT 1 VS. TREATMENT 2

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Treatment1	29.0000	1.7500	0.0603	0.0001
Treatment2	29.0000	1.7500	0.0603	0.0001

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	0.0000	1.0000	0.0000	0.0000	1.0000	4.0130
Within Groups	0.0037	56.0000	0.0001			
Total	0.0037	57.0000				

TREATMENT 2 VS. TREATMENT 3

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Treatment2	29.0000	1.7500	0.0603	0.0001
Treatment3	29.0000	1.7490	0.0603	0.0001

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	0.0000	1.0000	0.0000	0.0003	0.9871	4.0130
Within Groups	0.0037	56.0000	0.0001			
Total	0.0037	57.0000				

TABLE 4.12  
ANOVA FOR T1

TREATMENT 1 VS. TREATMENT 2

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Treatment1	28.0000	187.9049	6.7109	0.0002
Treatment2	28.0000	188.6560	6.7377	0.0002

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	0.0101	1.0000	0.0101	46.5678	0.0000	4.0195
Within Groups	0.0117	54.0000	0.0002			
Total	0.0218	55.0000				

TREATMENT 2 VS. TREATMENT 3

SUMMARY

<i>Groups</i>	<i>Count</i>	<i>Sum</i>	<i>Average</i>	<i>Variance</i>
Treatment2	28.0000	188.6560	6.7377	0.0002
Treatment3	28.0000	193.3441	6.9051	0.0002

ANOVA

<i>Source of Variation</i>	<i>SS</i>	<i>df</i>	<i>MS</i>	<i>F</i>	<i>P-value</i>	<i>F crit</i>
Between Groups	0.3925	1.0000	0.3925	1860.7613	0.0000	4.0195
Within Groups	0.0114	54.0000	0.0002			
Total	0.4039	55.0000				

## RESULTS FOR RESEARCH QUESTION 3: WORKLOAD SEQUENCE

### Effect of # Workload Units Processed

For each of the nine database operations, there was a strongly significant effect of the # Workload Units Processed on response time. The positive slope coefficients reported in Tables 4.13 - 4.21 are in seconds per workload unit and may be multiplied by 1000 to get the change per 1000 workload units.

### Effect of Dummy Variables Indicating Preceding Conditions

The magnitude of the coefficients for the dummy variables can be interpreted as the number of seconds response time would increase, on average, for an operation following Q8 or T1, over the response time expected for the same operation following any other operation.

In general, the presence of Q8 as the preceding operation caused a significant increase in response times for most operations. The exceptions were when Q8 preceded Q4, T1, and Q8 itself; there was no significant change in response time in these cases. Q4 also processed database segment 4, like Q8. T1 and Q8 were such I/O-intensive operations, that the cache was flushed regardless of preceding conditions.

When the preceding operation was T1, there was a significant increase in response times for all operations except T1 itself. (In this case T1 followed by another T1 resulted in a significant drop in response time.)



For operations which were the second one following either a Q8 or T1, however, there generally was not a significant effect on response time. The effect of these two operations seemed to be dampened later in the workload.

In most cases, when an operation followed itself, there was a significant decrease in response time, as one would expect from the warming of the cache.

All regression models had significant F statistics. For those operations with relatively small working sets compared to available RAM and client cache, there were moderate adjusted R Square values (0.35 - 0.50), indicating that this set of variables left quite a bit of variation unexplained (i.e., from the effects of other preceding operations not articulated in the model). For Q8 and T1, whose working set approximated or exceeded the memory constraints higher R Squares were obtained (0.88).

## FURTHER ANALYSIS OF VARIABILITY IN RESPONSE TIMES

For the smallest database (4/100/3/3) and the largest database (7/2003/3), response times for each database operation were analyzed graphically to gain more insight into the variability a user might observe when executing such a stochastic workload. A histogram illustrating the distribution of response times per operation was plotted. In addition, a linear graph of response times per operation in chronological order of execution was plotted.

As the sample graphs indicate for database 4/100/3/3, there was a wide variance in response times for each database operation, when the workload consisted of a continuous sequence of random operations. From a statistical process control point of view, there were multiple processes in action during the simulation. (Similar graphs for database 7/200/3/3 are in Appendix E )

#### Effect of the Seed Used to Generate Random Operation Stream

The stream of random operations was re-generated, using another seed, and this second stream was run against database 4/100/3/3. Response times for each operation, using both seeds, are compared in the following Figures.

It was difficult just to compare the average response time and standard deviation per database operation, which were skewed by initialization conditions for some (but not all) operations. It was more informative to look at the range of response times graphically for both seeds. Initialization conditions were left in the data in this analysis, because such variation might occur normally due to memory faults, if in real operation the processor were not solely dedicated to running just OO7 work.

#### Effect of Removing Q8 and T1 From the Workload

During Experiment 3, extremely long response times on the time series charts per database operation were examined visually to determine which operation, or pair of operations, had directly preceded them in the random sequence. With this particular

workload, Q8 and T1 had a tendency to be the preceding operations. This resulted in the inclusion of the dummy variables in the regression models for workload sequence.

In Table 4.8, note the sizes of Segments 4 and 6 of the physical database, in comparison to the client cache and RAM size on the implementation of the OO7 database in this study. More specifically, when Q8 executed, it flushed the client cache and left a working set that only one other operation in this simulated workload could use (Q4). T1, on the other hand, flushed the cache, but left a working set that several other operations in this workload could use.

To determine what response times would look like without Q8 and T1 in the workload mix, the same sequence of operations was run again, with these two operations removed from the sequence. (In practical terms, this would be equivalent to scheduling jobs in a production environment, perhaps running Q8- and T1-types of operations separately at controlled times.)

The following graphs illustrate the impact of removing just Q8, then removing both Q8 and T1, on the variability in Q1 response times on database 4/100/3/3. Again, similar results were obtained for both seeds in the random number generator.

TABLE 4.13  
Q1 WORKLOAD REGRESSION MODEL

<i>Regression Statistics</i>	
Multiple R	0.60
R Square	0.36
Adjusted R Square	0.35
Standard Error	2.00
Observations	392

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	6	871.17	145.20	36.29	8.39E-35
Residual	385	1540.34	4.00		
Total	391	2411.51			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	-1.51	0.38	-4.02	7.13E-05	-2.25	-0.77
#WkldUnits	3.47E-05	4.85E-06	7.15	4.43E-12	2.51E-05	4.42E-05
FollowsQ8	3.20	0.28	11.44	2.81E-26	2.65	3.75
TwoAfterQ8	0.39	0.29	1.35	0.18	-0.18	0.97
FollowsT1	1.93	0.46	4.19	3.42E-05	1.02	2.83
TwoAfterT1	-0.09	0.37	-0.25	0.80	-0.82	0.63
FollowsItself	-0.86	0.34	-2.55	0.01	-1.52	-0.20

TABLE 4.14  
Q2 WORKLOAD REGRESSION MODEL

<i>Regression Statistics</i>	
Multiple R	0.70
R Square	0.49
Adjusted R Square	0.48
Standard Error	6.27
Observations	357

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	6	13025.83	2170.97	55.27	8.23E-48
Residual	350	13748.29	39.28		
Total	356	26774.12			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	-12.80	1.23	-10.44	2.09E-22	-15.21	-10.39
#WkldUnits	2.51E-04	1.59E-05	15.73	1.56E-42	2.19E-04	2.82E-04
FollowsQ8	6.45	1.05	6.13	2.35E-09	4.38	8.52
TwoAfterQ8	3.44	1.15	3.00	2.92E-03	1.18	5.69
FollowsT1	4.89	0.95	5.13	4.80E-07	3.01	6.76
TwoAfterT1	1.24	1.15	1.08	0.28	-1.02	3.49
FollowsItself	-4.36	1.44	-3.02	2.67E-03	-7.19	-1.52

TABLE 4.15  
Q3 WORKLOAD REGRESSION MODEL

<i>Regression Statistics</i>	
Multiple R	0.83
R Square	0.69
Adjusted R Square	0.69
Standard Error	12.82
Observations	462

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	6	167374.92	27895.82	169.79	9.58E-113
Residual	455	74754.50	164.30		
Total	461	242129.42			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	-41.24	2.21	-18.69	2.98E-58	-45.57	-36.90
#WkldUnits	8.77E-04	2.86E-05	30.61	1.52E-112	8.20E-04	9.33E-04
FollowsQ8	11.78	2.02	5.84	1.02E-08	7.81	15.75
TwoAfterQ8	2.84	2.12	1.34	0.18	-1.33	7.01
FollowsT1	6.74	1.94	3.47	5.70E-04	2.92	10.56
TwoAfterT1	1.70	1.67	1.01	0.31	-1.59	4.98
FollowsItself	-6.52	1.57	-4.15	3.89E-05	-9.60	-3.43

TABLE 4.16  
Q4 WORKLOAD REGRESSION MODEL

<i>Regression Statistics</i>	
Multiple R	0.71
R Square	0.50
Adjusted R Square	0.49
Standard Error	5.84
Observations	322

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	6	10759.60	1793.27	52.50	1.24E-44
Residual	315	10759.81	34.16		
Total	321	21519.41			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	2.06	0.50	4.15	4.36E-05	1.08	3.04
#WkldUnits	5.63E-03	3.52E-04	15.98	1.63E-42	4.94E-03	6.32E-03
FollowsQ8	0.69	1.34	0.51	0.61	-1.96	3.33
TwoAfterQ8	-3.12	2.25	-1.39	0.17	-7.55	1.30
FollowsT1	4.80	1.18	4.05	6.35E-05	2.47	7.13
TwoAfterT1	2.67	1.08	2.48	0.01	0.55	4.78
FollowsItself	-5.03	1.00	-5.05	7.54E-07	-7.00	-3.07

TABLE 4.17  
Q5 WORKLOAD REGRESSION MODEL

<i>Regression Statistics</i>	
Multiple R	0.64
R Square	0.41
Adjusted R Square	0.41
Standard Error	2.47
Observations	504

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	6	2147.12	357.85	58.63	9.33E-55
Residual	497	3033.23	6.10		
Total	503	5180.35			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	-0.03	0.17	-0.18	0.86	-0.37	0.31
#WkldUnits	1.37E-03	1.19E-04	11.4940932	2.82E-27	1.13E-03	1.60E-03
FollowsQ8	5.83	0.49	11.88	7.80E-29	4.87	6.80
TwoAfterQ8	0.44	0.44	1.01	0.32	-0.42	1.30
FollowsT1	2.73	0.36	7.55	2.15E-13	2.02	3.44
TwoAfterT1	0.83	0.32	2.57	0.01	0.20	1.47
FollowsItself	-0.67	0.27	-2.47	0.01	-1.21	-0.14



TABLE 4.18  
Q6 WORKLOAD REGRESSION MODEL

<i>Regression Statistics</i>	
Multiple R	0.66
R Square	0.44
Adjusted R Square	0.43
Standard Error	2.21
Observations	308

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	6	1143.99	190.67	39.14	4.55E-35
Residual	301	1466.27	4.87		
Total	307	2610.26			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	0.20	0.18	1.12	0.26	-0.15	0.56
#WkldUnits	7.59E-04	9.07E-05	8.37	2.23E-15	5.80E-04	9.37E-04
FollowsQ8	4.63	0.40	11.57	7.32E-26	3.84	5.42
TwoAfterQ8	1.09	0.37	2.95	3.45E-03	0.36	1.82
FollowsT1	2.78	0.61	4.56	7.60E-06	1.58	3.98
TwoAfterT1	-0.49	0.61	-0.81	0.42	-1.69	0.71
FollowsItself	-0.52	0.85	-0.62	0.54	-2.19	1.14

TABLE 4.19  
Q8 WORKLOAD REGRESSION MODEL

<i>Regression Statistics</i>	
Multiple R	0.94
R Square	0.89
Adjusted R Square	0.88
Standard Error	16.70
Observations	329

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	5	699621.01	139924.20	501.93	6.64E-150
Residual	323	90042.95	278.77		
Total	328	789663.96			

	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	-58.11	3.35	-17.37	3.36E-48	-64.69	-51.53
#WkldUnits	2.20E-03	4.42E-05	49.88	8.46E-154	2.12E-03	2.29E-03
FollowsQ8	-1.22	3.81	-0.32	0.75	-8.71	6.27
TwoAfterQ8	-3.17	3.38	-0.94	0.35	-9.81	3.47
FollowsT1	11.86	3.38	3.51	5.06E-04	5.22	18.50
TwoAfterT1	10.21	3.07	3.33	9.77E-04	4.17	16.25

TABLE 4.20  
T1 WORKLOAD REGRESSION MODEL

<i>Regression Statistics</i>	
Multiple R	0.94
R Square	0.88
Adjusted R Square	0.88
Standard Error	39.23
Observations	343

ANOVA

	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	5	3800984.13	760196.83	494.01	1.03E-152
Residual	337	518583.34	1538.82		
Total	342	4319567.47			

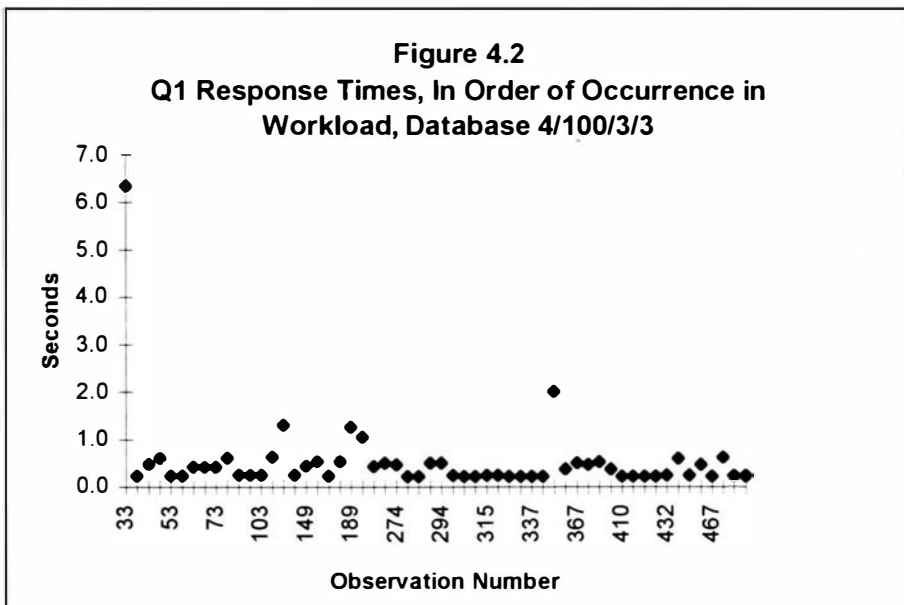
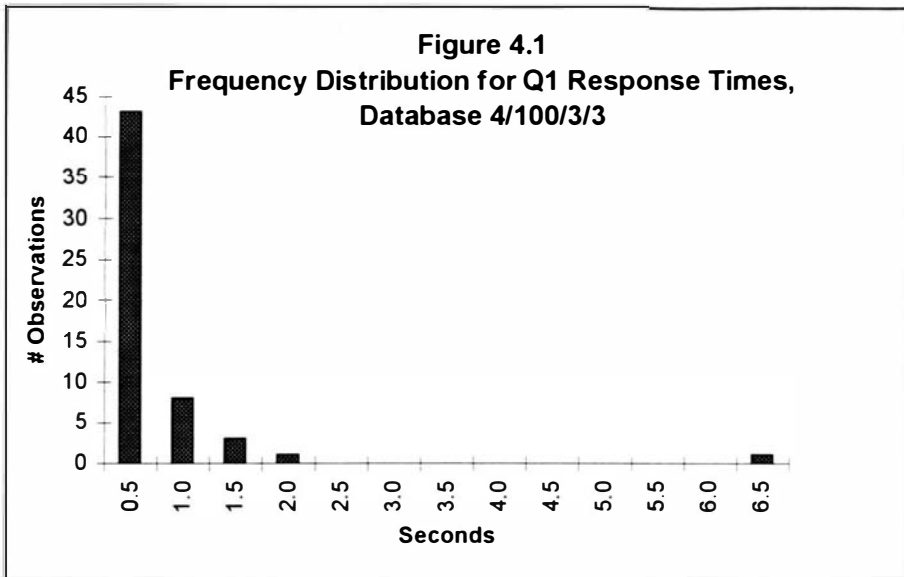
	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	-2.35	3.05	-0.77	0.44	-8.35	3.64
#WkldUnits	8.81E-04	1.78E-05	49.62	5.72E-157	8.46E-04	9.16E-04
FollowsQ8	12.38	8.10	1.53	0.13	-3.54	28.31
TwoAfterQ8	0.02	9.38	0.00	1.00	-18.43	18.48
FollowsT1	-13.10	7.26	-1.80	0.07	-27.38	1.18
TwoAfterT1	-10.58	6.16	-1.72	0.09	-22.70	1.55

TABLE 4.21  
T6 WORKLOAD REGRESSION MODEL

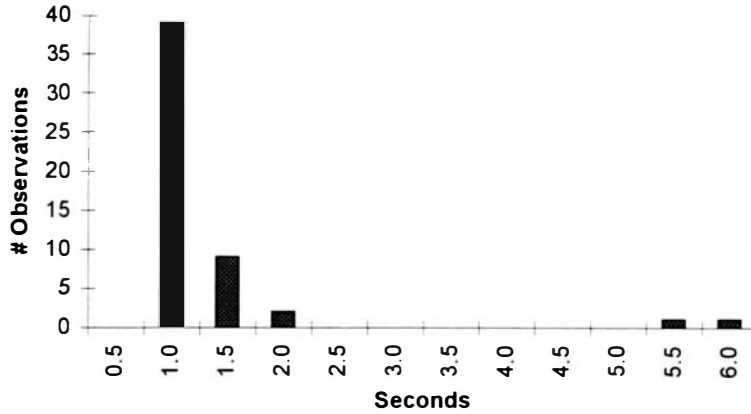
<i>Regression Statistics</i>	
Multiple R	0.64
R Square	0.41
Adjusted R Square	0.40
Standard Error	2.69
Observations	378

ANOVA					
	<i>df</i>	<i>SS</i>	<i>MS</i>	<i>F</i>	<i>Significance F</i>
Regression	6	1862.71	310.45	42.97	9.25E-40
Residual	371	2680.35	7.22		
Total	377	4543.06			

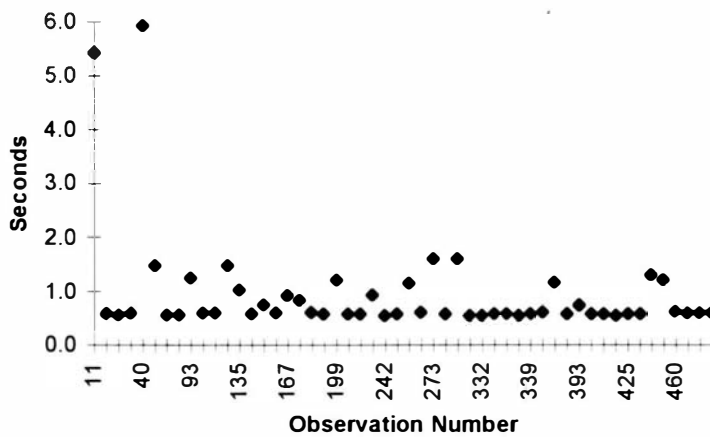
	<i>Coefficients</i>	<i>Standard Error</i>	<i>t Stat</i>	<i>P-value</i>	<i>Lower 95%</i>	<i>Upper 95%</i>
Intercept	0.04	0.22	0.20	0.84	-0.39	0.48
#WkldUnits	6.75E-04	5.98E-05	11.2767885	1.47E-25	5.57E-04	7.92E-04
FollowsQ8	4.28	0.49	8.67	1.37E-16	3.31	5.25
TwoAfterQ8	1.49	0.40	3.71	2.37E-04	0.70	2.28
FollowsT1	3.05	0.43	7.05	8.96E-12	2.20	3.90
TwoAfterT1	-0.51	1.09	-0.47	0.64	-2.65	1.63
FollowsItself	-0.84	0.42	-1.98	0.05	-1.67	-4.83E-03



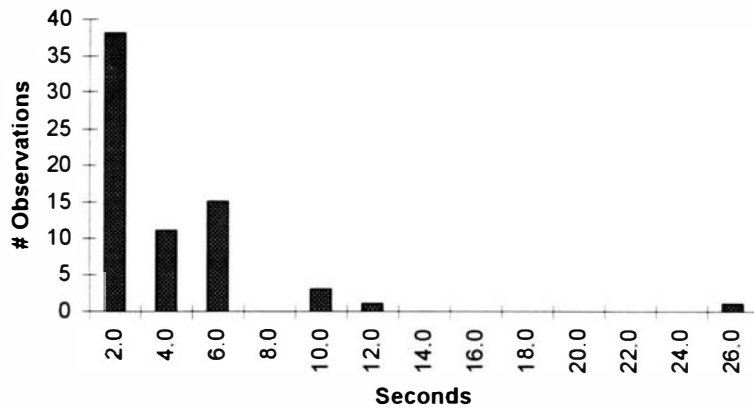
**Figure 4.3**  
**Frequency Distribution for Q2 Response Times,**  
**Database 4/100/3/3**



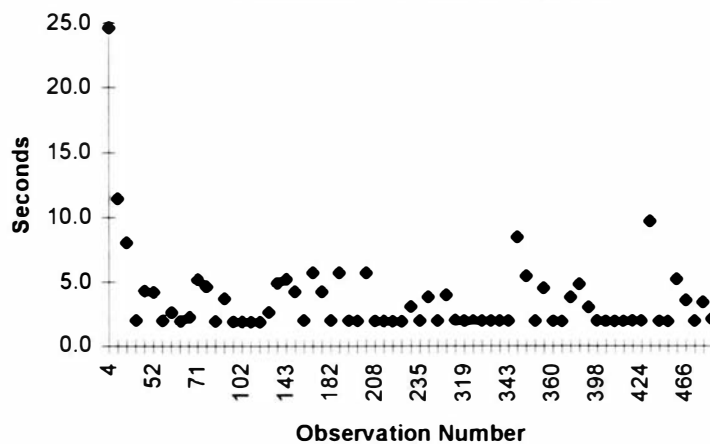
**Figure 4.4**  
**Q2 Response Times, In Order of Occurrence in**  
**Workload, Database 4/100/3/3**



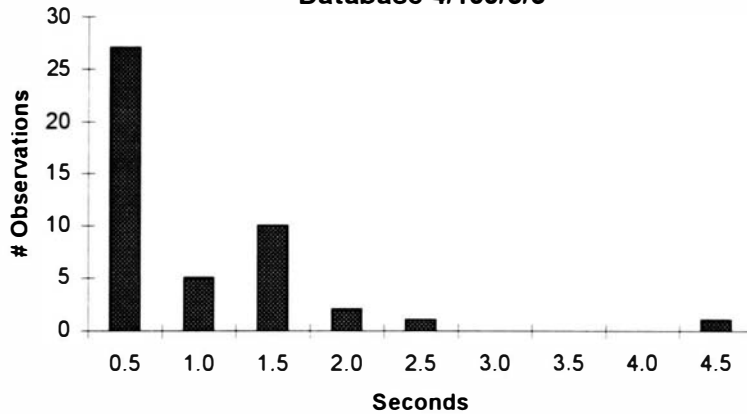
**Figure 4.5**  
**Frequency Distribution for Q3 Response Times,**  
**Database 4/100/3/3**



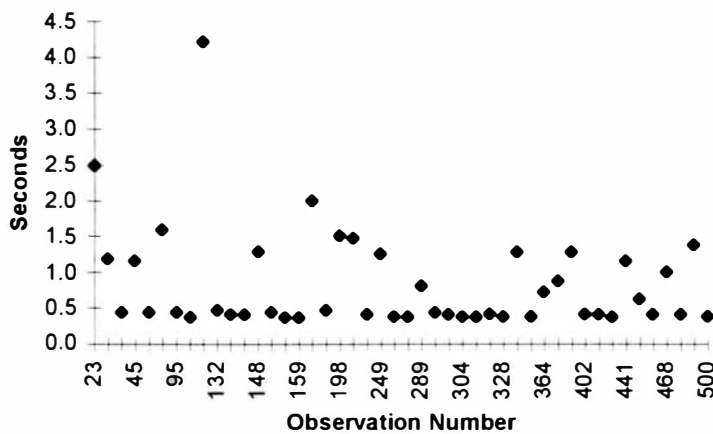
**Figure 4.6**  
**Q3 Response Times, In Order of Occurrence in**  
**Workload, Database 4/100/3/3**



**Figure 4.7**  
**Frequency Distribution for Q4 Response Times,**  
**Database 4/100/3/3**

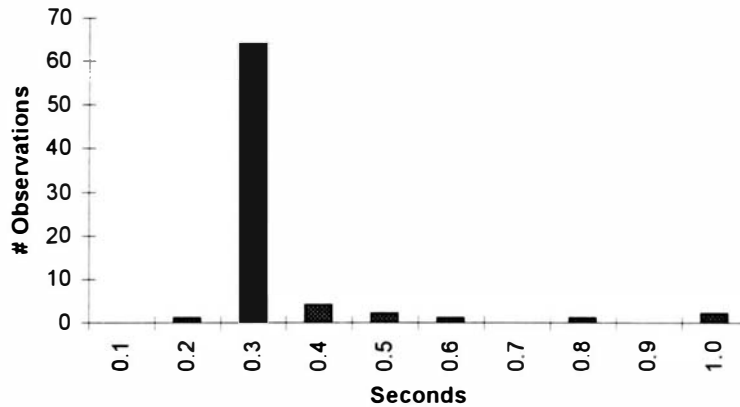


**Figure 4.8**  
**Q4 Response Times, In Order of Occurrence in**  
**Workload, Database 4/100/3/3**

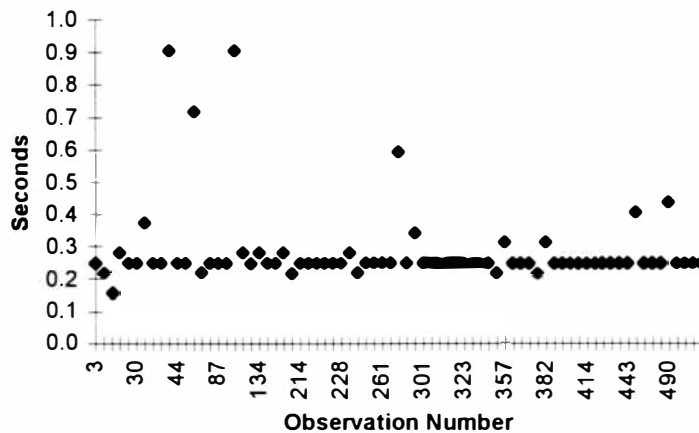




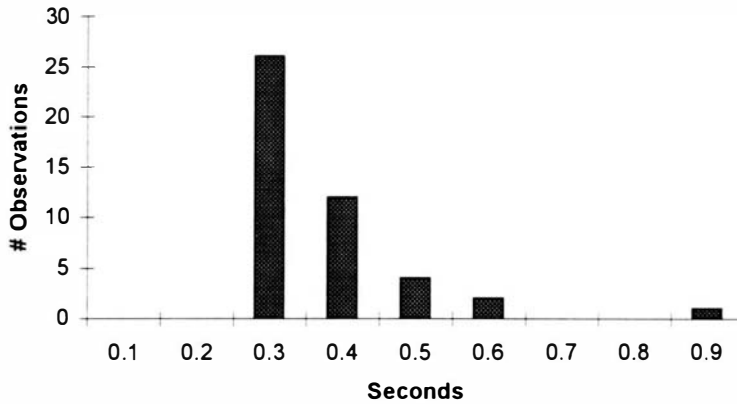
**Figure 4.9**  
**Frequency Distribution for Q5 Response Times,**  
**Database 4/100/3/3**



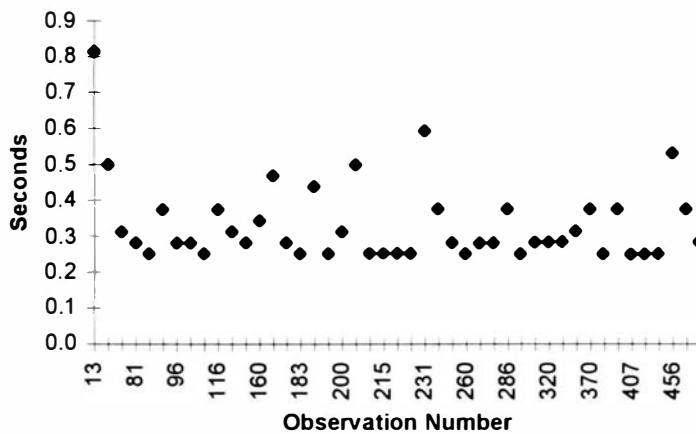
**Figure 4.10**  
**Q5 Response Times, In Order of Occurrence in**  
**Workload, Database 4/100/3/3**



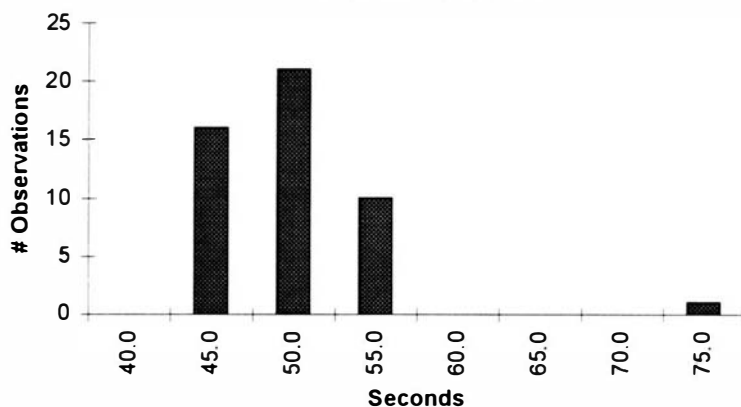
**Figure 4.11**  
**Frequency Distribution for Q6 Response Times,**  
**Database 4/100/3/3**



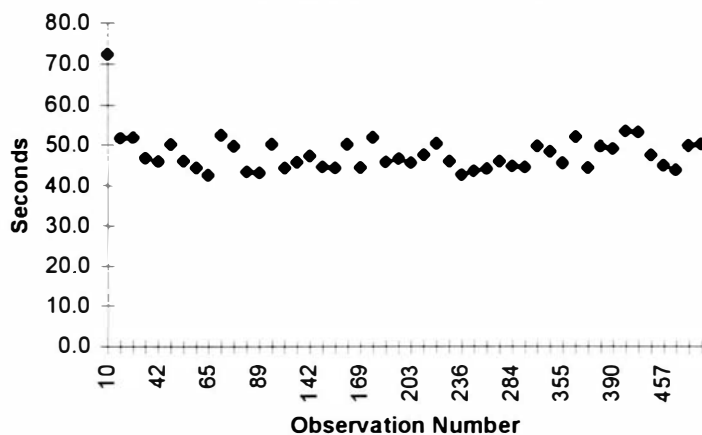
**Figure 4.12**  
**Q6 Response Times, In Order of Occurrence in**  
**Workload, Database 4/100/3/3**



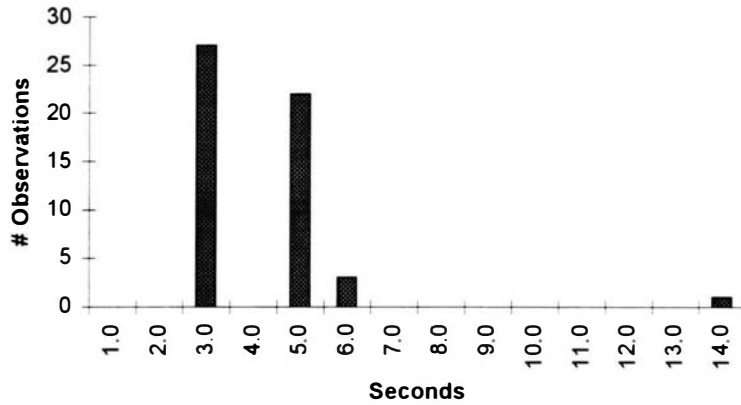
**Figure 4.13**  
**Frequency Distribution for Q8 Response Times,**  
**Database 4/100/3/3**



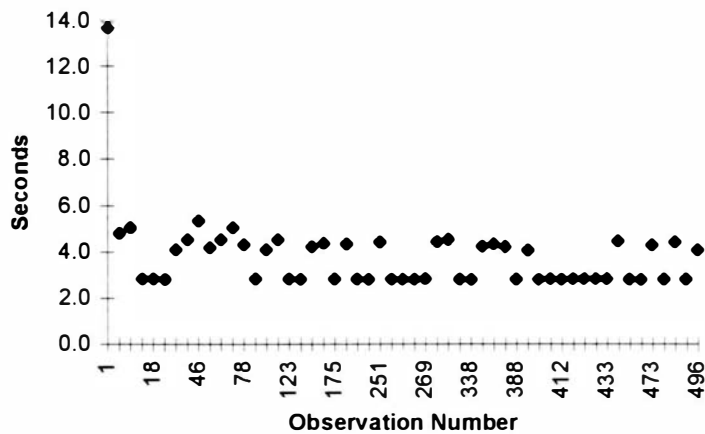
**Figure 4.14**  
**Q8 Response Times, In Order of Occurrence in**  
**Workload, Database 4/100/3/3**



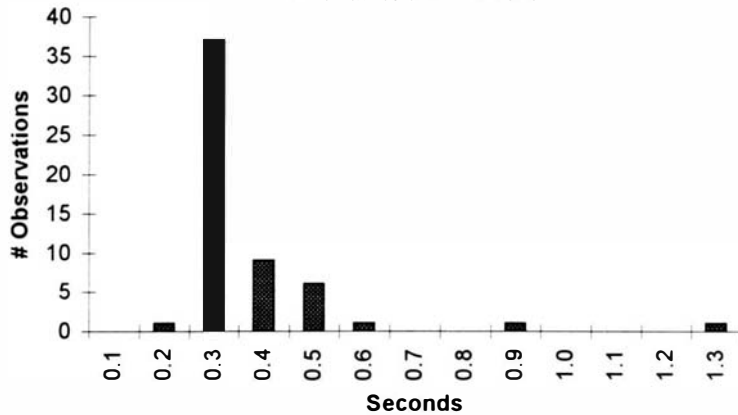
**Figure 4.15**  
**Frequency Distribution for T1 Response Times,**  
**Database 4/100/3/3**



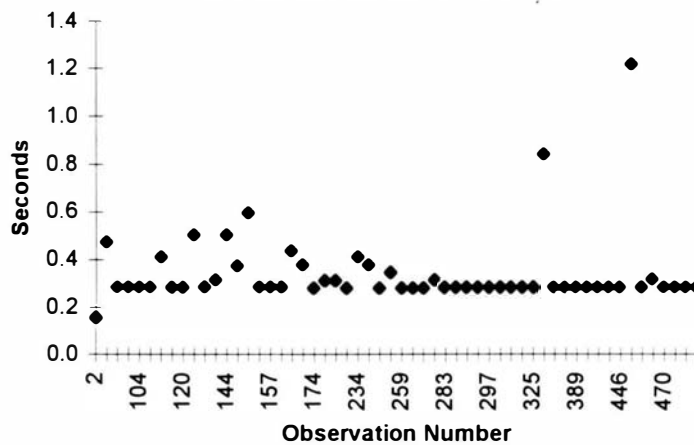
**Figure 4.16**  
**T1 Response Times, In Order of Occurrence in**  
**Workload, Database 4/100/3/3**

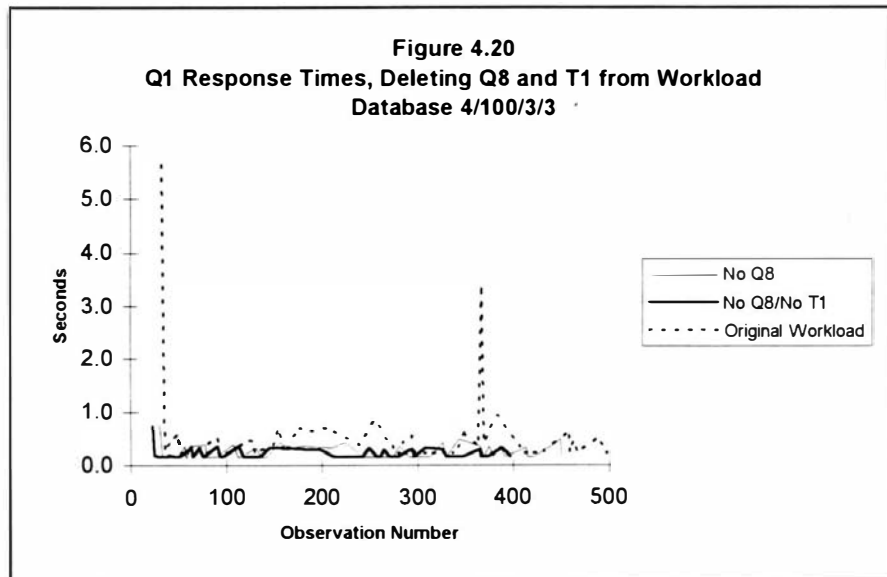
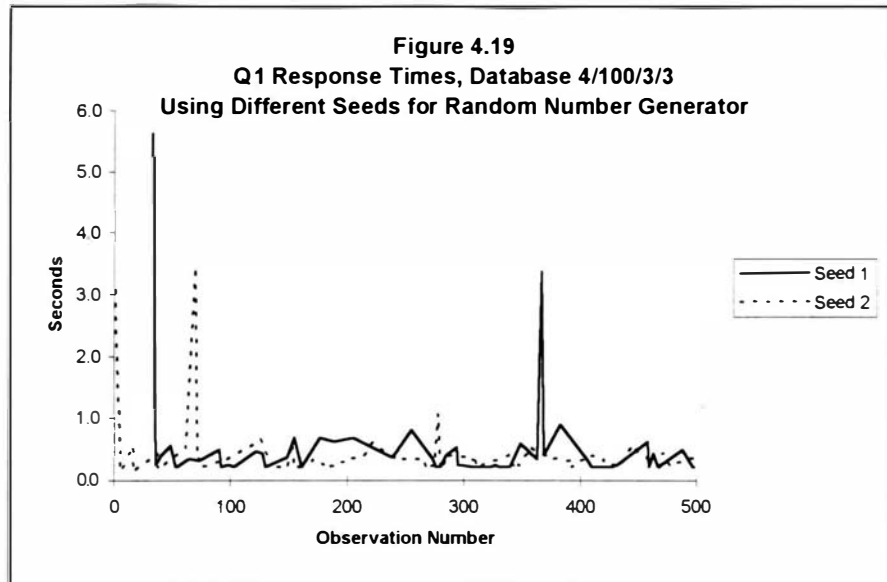


**Figure 4.17**  
**Frequency Distribution for T6 Response Times,**  
**Database 4/100/3/3**



**Figure 4.18**  
**T6 Response Times, In Order of Occurrence in**  
**Workload, Database 4/100/3/3**





## CHAPTER V

### CONCLUSIONS

#### SUMMARY OF RESULTS

In this research, several logical design characteristics of the OO7 Benchmark database application were manipulated experimentally to determine the effect on response time to perform selected OO7 queries and traversals. In Experiments 1 and 2, the degree of aggregation and the degree of inheritance were increased systematically to see whether there was an impact on response times. In Experiment 3, operations were executed in a randomly-generated order to determine the variability in response times a user would experience in a continuous, dynamic workload.

#### Effect of the Degree of Aggregation on Response Time

For a given query or traversal, for a given clustering policy and fetch policy, response time was linear with respect to the number of “part-of” relationship objects processed. When the working set of a query or traversal approached or exceeded memory constraints for RAM or for the ObjectStore client disk cache, response time had a different, but still linear relationship with the number of relationship objects processed.

### Effect of the Degree of Inheritance on Response Time

The effect of increasing the degree of inheritance in the class definitions and in the execution of virtual functions was statistically significant only for T1 operations, which had highly repetitive processing of objects with inherited attributes and virtual functions. However, the difference in mean response times between treatment levels was not large enough to be important.

### Effect of the Sequence of Operations on Response Time

In this study, the most dramatic influence on response times for all operations was the sequence of operations in the stochastic workload. Unlike results in a benchmarking type of study, response times for each operation exhibited a high degree of variability, depending on the preceding operations and the working set left in the client cache and main memory for the next transaction.

Even the most sensible clustering policy used to generate the physical database, based on the most likely transaction workload, would not optimize performance if the transaction workload were highly variable. In this study, only nine types of operations were simulated. In multi-user and other single-user workloads, there could be even more variety, causing more memory faults.



## IMPLICATIONS FOR APPLICATION DESIGNERS

As always, a thorough job is necessary in specifying logical database requirements, projected cardinalities for database objects, and most common and/or most resource-intensive queries and traversals. During physical database design, clustering policy and projected segment sizes need to be developed, including ObjectStore overhead.

Relationships and index paths which overlap segments need to be examined for possible impact on transactions involving multiple segments. Any segments which will be processed sequentially and in full need to have sizes which will fit in the client cache and RAM. Application designers need to be aware of projected constraints on hard disk space (which would affect the setting for the client cache and the availability of free space for the OS/2 swap file) as well as RAM.

The above decisions are made and implemented one time, or occasionally as the database schema evolves, in the static database physical structure. It would be impractical to change the database structure frequently. Other design decisions can also affect the dynamic behavior of the application. First, the fetch policy can be set for a transaction or set of transactions, to balance between pulling in data as efficiently as possible, without flushing the client cache unnecessarily. In addition to capitalizing on locality within a given transaction, or a short-term sequence of transactions, system designers need to consider a scheduling policy for transactions that cause an unacceptable level variability in response times. In order to do this effectively, the average response time and potential

variance need to be projected for the most important transactions--the ones that take the most system resources. Then a decision needs to be made on what constitutes unacceptably high response times, and design decisions made accordingly.

## CONTRIBUTIONS AND LIMITATIONS OF STUDY

The major contributions of this study are: 1) a valid independent variable for predicting response times as a function of the number of relationship objects, generally-applicable to any ObjectStore database application with aggregation relationships, 2) evidence that while degree of inheritance can change response times in some cases, this change is not a serious problem in databases of moderate size, and 3) a methodology for characterizing a potential workload for an object-oriented database application, that is at a higher logical level and provides a more realistic picture than a benchmarking study.

Similar experiments could be implemented with a roughly designed application during physical design, by including the most important object classes and the most critical transaction types. Decisions can then be made on clustering and segment layouts, transaction scheduling, RAM requirements, client cache setting, and fetch policy.

Another major contribution of this study is a clear picture of the inherent variability in response times in a continuous stream of transactions. In those situations where the variability is unacceptable to the user, the assignable causes must be determined and either removed or modified (e.g., scheduling of critical transactions) to improve the performance capability of the system.

The results in this study are a direct function of the particular OO7 operations tested. Operations that update the physical database may cause further variability in response times, and workloads dominated by certain transactions may show a different variability. The simulation methodology used in this study could certainly be expanded to take these conditions into account.

## FUTURE WORK

It would be desirable to extend this work to other single-user application areas, to see if rough mockup physical designs can be implemented in a cost-effective manner to estimate average response times and potential variability in a projected workload. This approach should also be extended to a multiuser workload, first with multiple users sharing a single application, and then with multiple users sharing a server with multiple applications.

## BIBLIOGRAPHY

Anderson, T., Berre, A., Mallison, M., Porter, H., and Schneider, B. The HyperModel Benchmark. In Proceedings Conference on Extending Database Technology (Venice, March 1990), Springer-Verlag Lecture Notes 416.

Anon. et al. A Measure of Transaction Processing Power. Datamation 31, 7 (April, 1985), 112-118.

Banks, J. and Carson, J. S. Discrete-Event System Simulation, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1984.

Berre, A. J. and Anderson, T. L. The HyperModel Benchmark for Evaluating Object-Oriented Databases. In Object-Oriented Databases With Applications to CASE, Networks, and VLSI CAD, 1991.

Blaser, A., Ed., Database Systems of the 90's, International Symposium, Muggelsee, Berlin, FRG, November 5-7, 1990, Proceedings, Berlin: Springer-Verlag, 1990.

Brownsmith, J. D. IBM Santa Teresa Laboratory, San Jose, California. Internal Communication, April 19, 1995.

Brumfield, J. A., Miller, J. L., and Chou, H. Performance Modeling of Distributed Object-Oriented Database Systems. In Proceedings Symposium on Databases in Parallel and Distributed Systems (Austin, Texas, December 5-7, 1988), 22-32.

Byte, State of the Art Section, "Objects for End Users", Vol. 17, No. 14, December, 1992.

Carey, M. J., DeWitt, D. J., and Naughton, J. F. The OO7 Benchmark. In Proceedings of the 1993 ACM SIGMOD Conference on the Management of Data (Washington, D.C., May, 1993), 12-21. Also available as Technical Report No. 1140, Computer Sciences Department, University of Wisconsin, revised January, 1994.

Carey, M. J., DeWitt, D. J., Kant, C., and Naughton, J. F. A Status Report on the OO7 OODBMS Benchmarking Effort. SIGPLAN Notices 29, 10 (October, 1994), 414-426.

Cattell, R. G. G. Object-Oriented DBMS Performance Measurement. In Advances in Object-Oriented Database Systems (Bad Munster am Stein-Ebernburg, FRG, September 27-30, 1988), Springer-Verlag.

Cattell, R. G. G. Object Data Management: Object-Oriented and Extended Relational Database Systems, 2nd Ed. Reading, Massachusetts: Addison-Wesley Publishing Company, 1994.

Cattell, R. G. G. and Skeen, J. Object Operations Benchmark. ACM Transactions on Database Systems 17, 1 (March 1992), 1-31.

Cesarini, F. and Salza, S., Eds., Database Machine Performance: Modeling Methodologies and Evaluation Strategies, Vol. 257 of Lecture Notes in Computer Science, Springer-Verlag, 1987.

Chaudhri, A. B. An Annotated Bibliography of Benchmarks for Object Databases. SIGMOD Record 24, 1 (March, 1995), 50-57.

Chidamber, S. R. and Kemerer, C. F. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, 20, 6 (June, 1994), 476- 493.

Datapro Information Services Group, Datapro Reports on UNIX Systems & Software, Delran, New Jersey: McGraw-Hill, October, 1992.

Date, C. J. An Introduction to Database Systems, Reading, Massachusetts: Addison-Wesley Publishing Company, 1990.

Dayal, U. and Traiger, I., Eds., Proceedings of Association for Computing Machinery Special Interest Group on Management of Data, 1987 Annual Conference, San Francisco, May 27-29, 1987, SIGMOD Record 16, 3 (December, 1987).

Dillon, W. R. and Goldstein, M. Multivariate Analysis: Methods and Applications. New York: John Wiley & Sons, Inc., 1984.

Dittrich, K. R., Ed., Advances in Object-Oriented Database Systems, Vol. 334 of Lecture Notes in Computer Science, Proceedings 2nd International Workshop on Object-Oriented Database Systems (Bad Munster am Stein-Ebernburg, FRG, September 27-30, 1988), Springer-Verlag.

Dittrich, K. R., Dayal, U., and Buchmann, A. P., Eds., On Object-Oriented Database Systems, Springer-Verlag, 1991.

Duhl, J. and Damon, C. A Performance Comparison of Object and Relational Databases Using the Sun Benchmark. In Proceedings of the ACM OOPSLA Conference (San Diego, California, September, 1988), 153-163.

Ferrari, D., Serazzi, G., and Zeigner, A. Measurement and Tuning of Computer Systems, Englewood Cliffs, NJ: Prentice-Hall, 1983.

Gray, J., Ed., The Benchmark Handbook for Database and Transaction Processing Systems, Morgan-Kaufmann, 1991.

Heidelberger, P., and Lavenberg, S. S. Computer Performance Evaluation Methodology. IEEE Transactions on Computers c-33, 12 (December 1984).

Hull, R. B., Morrison, R., and Stemple, D. W., Eds., Proceedings of the Second International Workshop on Database Programming Languages, Gleneden Beach, Oregon, June 4-8, 1989, San Mateo, California: Morgan Kaufmann Publishers, Inc., 1989.

Hurson, A. R., Pakzad, S. H., and Cheng, J. Object-Oriented Database Management Systems: Evolution and Performance Issues. Computer 26, 2 (February 1993), 48-60.

Jajodia, S., Kim, W., and Silberschatz, A., Eds., Proceedings International Symposium on Databases in Parallel and Distributed Systems, Austin, Texas, December 5-7, 1988, Washington: IEEE Computer Society Press, 1988.

Kim, W. and Lochovsky, F. H., Eds., Object-Oriented Concepts, Databases, and Applications, Reading, Massachusetts: Addison-Wesley Publishing Company, 1989.

Kim, W. Introduction to Object-Oriented Databases, Cambridge, Mass.: MIT Press, 1990.

Kim, W., Ballou, N., Garza, J. F., and Woelk, D. A Distributed Object-Oriented Database System Supporting Shared and Private Databases. ACM Transactions on Information Systems 9, 1, (January 1991), 31-51.

Maier, D. Making Database Systems Fast Enough for CAD Applications. In Object-Oriented Concepts, Databases, and Applications, 1989.

Melton, K. I. Introduction to Statistics for Process Studies, 2nd Ed., New York: McGraw-Hill, 1993.

Rubenstein, W. B., Kubicar, M. S., and Cattell, R. G. G. Benchmarking Simple Database Operations. In Proceedings of Association for Computing Machinery Special Interest Group on Management of Data, 1987 Annual Conference (San Francisco, May 27-29, 1987), SIGMOD Record, 16, 3 (December, 1987), 387-394.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W. Object-Oriented Modeling and Design, Englewood Cliffs, N.J.: Prentice Hall, 1991.

White, S. J. and DeWitt, D. J. Implementing Crash Recovery in QuickStore: A Performance Study. SIGMOD Record, 24, 2 (June, 1995), 187-198.

APPENDIX A  
PROGRAM FOR GENERATING DATABASE OPERATIONS  
IN PSEUDORANDOM ORDER

```
// E. M. Walk 8/95
// randopns.cpp
// Program for generating database operations in pseudorandom order

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

long seed = 1;      //could be anything between 1 and 2147483646
//keep a cumulative count for the number of times an operation is called
//initialize all cumulative counts for operations to zero
int countT1 = 0;
int countT6 = 0;
int countQ1 = 0;
int countQ2 = 0;
int countQ3 = 0;
int countQ4 = 0;
int countQ5 = 0;
int countQ6 = 0;
int countQ8 = 0;

FILE *fp;
float RandomNum (long seed);
void RandomOpns (float R);

int main (void)
{
    int n;
    float randnumber;
    if((fp = fopen("a:randopns.dat", "w"))==NULL){
        printf("Cannot open file\n");
        exit(1);
    }
    for (n=1;n<=500;n++) {
        printf("%3i ",n);
        randnumber = RandomNum(seed);
        RandomOpns(randnumber);
    } /* endfor */
    fclose(fp);
}
```



```

    return 0;
}

float RandomNum (long s)
//Random number generator using multiplicative linear congruential algorithm
//Implementation from Parks and Miller, CACM 31(10) 10/88
//Ported to IBM C/C++ FirstStep compiler Intel 486 E. M. Walk 8/95
{
    int a = 16807;           //multiplier
    long m = 2147483647;     //modulus
    long q = 127773;        //m div a
    int r = 2836;           //m mod a
    long lo, hi, test;
    float result;

    hi = s / q;             //div
    lo = s % q;             //mod
    test = a * lo - r * hi;
    if (test > 0) seed = test;
    else seed = test + m;
    result = (float) seed / (float) m;
    //printf("seed = %i result = %f ",seed,result); //to debug
    return (result);
}

void RandomOps (float R)
//generate random variate whichOp from the following discrete distribution
// whichOp relat. freq. cumul. freq.
//  T1   .11      .11
//  T6   .11      .22
//  Q1   .12      .34
//  Q2   .11      .45
//  Q3   .11      .56
//  Q4   .11      .67
//  Q5   .11      .78
//  Q6   .11      .89
//  Q8   .11      1.00
{
    int count;
    char whichOp[7];

    if (R <= .11) { strcpy(whichOp,"T1"); count = ++countT1;}
    else if (R <= .22) { strcpy(whichOp,"T6"); count = ++countT6;}

```

```

else if (R <= .34) { strcpy(whichOp,"Q1"); count = ++countQ1;}
else if (R <= .45) { strcpy(whichOp,"Q2"); count = ++countQ2;}
else if (R <= .56) { strcpy(whichOp,"Q3"); count = ++countQ3;}
else if (R <= .67) { strcpy(whichOp,"Q4"); count = ++countQ4;}
else if (R <= .78) { strcpy(whichOp,"Q5"); count = ++countQ5;}
else if (R <= .89) { strcpy(whichOp,"Q6"); count = ++countQ6;}
else if (R <= 1.00) { strcpy(whichOp,"Q8"); count = ++countQ8;}

```

```

printf("whichOp = %-10s Count = %3i\n",whichOp,count);

```

```

//save whichOp to a file
fprintf(fp, "%s\n",whichOp);
}

```

## APPENDIX B

### OO7 BENCHMARK, MODIFIED FOR SIMULATED TRANSACTION WORKLOAD

```
//OO7 bench.cpp revised by E. M. Walk 8/95 as simulate.cpp
//  Simulate.cpp opens a config file for the database to be studied, reads in an operation
//from a file of randomly-generated operations, runs the benchmark operation using OO7
//benchmark code, writes results to an output file and to the screen, repeats for a user-
//specified number of operations
//  OO7 bench.cpp was part of OO7 benchmark jointly developed by Carey, Dewitt, and
//Naughton at Univ. of Wisconsin and developers at Object Design, Inc., 1993 for
//ObjectStore Release 2.0.1.
//  OO7 was ported to ObjectStore 3.1 for OS/2 in 1995 by J. Brownsmith, IBM Santa
//Teresa Laboratory, and enhanced with optional operations and options for
//environmental variables. Because of changes this program is not comparable to the
//Univ. of Wisconsin version. The ported version was offered as-is, with no warranty
//as to correctness or quality. The use of parts of this benchmark for this dissertation are
//gratefully acknowledged.
```

```
#include <fstream.h>                                //added emw 8/95
#include <string.h>
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#include <iostream.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include <ostore/relat.hh>
#include "OO7.h"
#include "GenParam.h"
#include "BenchPar.h"
#include "VarParam.h"
#include "baidlist.h"
```

```
extern int RealWork;      // set to one to make DoNothings do work.
extern int WorkAmount;    // controls how much work DoNothings do
extern void SetParams(char* configFileName);
extern int traverse7();
extern int query1();
extern int query1a();
extern int query1b(const os_coll_query & q1b);
extern int query2();
extern int query3();
```

```

extern int  query4();
extern int  query4a();
extern int  query5();
extern int  query6();
extern int  query7();
extern int  query8();
extern int  query8a();
extern int  query8b();
extern int  query8c();
extern int  query8d();
extern void insert1();
extern void delete1();
extern int  reorg1();
extern int  reorg2();

////////////////////////////////////
//
// Global Variables, etc., for Benchmarking ODB Operations
//
////////////////////////////////////

os_database* oo7db;

os_Set<AtomicPart*>*  AtomicPart_extent = 0;
os_Set<BaseAssembly*>* BaseAssembly_extent = 0;
os_Set<CompositePart*>* CompositePart_extent = 0;
os_Set<Document*>*    Document_extent = 0;
os_Set<Module*>*      Module_extent = 0;

int  nextAtomicId = 0;
int  nextCompositeId = 0;
int  nextComplexAssemblyId = 0;
int  nextBaseAssemblyId = 0;
int  nextModuleId = TotalModules;
int  debugMode = FALSE;

float startWallTime;
float endWallTime;
float startWarmTime;

char  *types[NumTypes] = {
    "type000", "type001", "type002", "type003", "type004",
    "type005", "type006", "type007", "type008", "type009"

```

```

    };

// here only to keep the make happy. used in gendb
// and in Insert!!!
BAIdList* private_cp;
BAIdList* shared_cp;

// os_typespec for persistent new
os_typespec *ComplexAssembly_type = new os_typespec("ComplexAssembly");
os_typespec *Document_type = new os_typespec("Document");
os_typespec *char_type = new os_typespec("char");
os_typespec *Module_type = new os_typespec("Module");
os_typespec *CompositePart_type = new os_typespec("CompositePart");
os_typespec *Manual_type = new os_typespec("Manual");
os_typespec *AtomicPart_type = new os_typespec("AtomicPart");
os_typespec *Connection_type = new os_typespec("Connection");
os_typespec *BaseAssembly_type = new os_typespec("BaseAssembly");

/* emw 8/95 ***** */
/* Replaced ParseCommandLine completely with file i/o */
/* See bench.cpp for this fcn */
////////////////////////////////////
// ParseCommandLine parses the original shell call to "bench", determining
// which operation to run, how many times to run it, and whether the
// individual runs are distinct transactions or are lumped together
// into one large transaction.
////////////////////////////////////

main(int argc, char** argv)
{

/* emw 8/95 added for file i/o ***** */
    if (argc != 4) {
        cout << "Usage: argv[0] <config filename><random opns filename><results filename>\n";
        return 1;
    } /* endif */

    ifstream inconfig(argv[1]); //input file
    ifstream inopns(argv[2]); //input file
    ofstream out(argv[3]); //output file

```

```

if (!inconfig || !inopns) {
    cout << "Cannot open input file.\n";
    return 1;
} /* endif */
if (!out) {
    cout << "Cannot open output file.\n";
    return 1;
} /* endif */
/* emw 8/95 end of new code ***** */

char resultText[200]; // to hold result message in order to avoid
                    // printf in timing section.
os_transaction* xact;

////////////////////////////////////
//
// Initialize
//
////////////////////////////////////

// initialize parameters for benchmark.
SetParams(argv[1]);

// need to malloc space for these, but don't need to store
// any info since Insert looks up assemblies "by hand".
shared_cp = new BAIdList[TotalCompParts+NumNewCompParts+1];
private_cp = new BAIdList[TotalCompParts+NumNewCompParts+1];

//objectstore::set_cache_size(CachePages * 8192);
//printf("Initialized system with call objectstore::set_cache_size(%d)\n",
//      CachePages * 8192);

char buf[12] = "=====";
cout << buf << " begin system parameters " << buf << buf << endl;
cout << "Using " << objectstore::release_name() << endl;
cout << "Page size is " << objectstore::get_page_size() << endl;
objectstore::set_cache_size(CachePages * 8192);
cout << "Initialized system with call objectstore::" <<
    "set_cache_size(" << CachePages*8192 << ")" << endl;
cout << "(" << CachePages*2 << objectstore::get_page_size()
    << ") pages" << endl;

```

```

int force_full_init = 1; // 0=partial (default); 1=full initialize
objectstore::initialize(force_full_init);
if (force_full_init)
    cout << "ObjectStore initialization procedures - all done" << endl;
else
    cout << "ObjectStore initialization procedures - some deferred" << endl;
if ( objectstore::get_opt_cache_lock_mode() )
{
    cout << "opt_cache_lock_mode is OFF" << endl;
} else {
    cout << "opt_cache_lock_mode is ON - "
        << " lock upgrade is optimized" << endl;
}

if ( objectstore::get_auto_open_read_whole_segment_mode() )
{
    cout << "read_whole_segment mode is ON" << endl;
} else {
    cout << "read_whole_segment mode is OFF" << endl;
}

if ( objectstore::get_auto_open_read_only_mode() )
{
    cout << "automatically-opened database read_only mode is ON" << endl;
} else {
    cout << "automatically-opened database read_only mode is OFF" << endl;
}

cout << "Write lock timeout is " <<
    objectstore::get_writelock_timeout <<
    " (milliseconds)" << endl;

//-----
// conditionally enable counters and hooks

// enable counters
if ((os_boolean)getenv("OS_COUNTERS"))
{
    objectstore::reset_counters();
    cout << "OS_COUNTERS are RESET to zero" << endl;
} else {
    cout << "OS_COUNTERS are NOT reset to zero" << endl;
}

```

```

// enable Release 3.1 realtime counters
if ((os_boolean)getenv("OS_RTCOUNTERS"))
{
    objectstore::record_realtime_counters(TRUE);
    cout << "OS_RTCOUNTERS are ENABLED" << endl;
} else {
    cout << "OS_RTCOUNTERS are NOT enabled" << endl;
}

// enable event hooks
if ((os_boolean)getenv("OS_HOOKS"))
{
    objectstore::enable_event_hooks();
    objectstore::set_default_hooks(); // all hooks
    cout << "OS_HOOKS are ENABLED" << endl;
} else {
    cout << "OS_HOOKS are NOT enabled" << endl;
}

// enable Release 3.1 OS/2 Toggle mapaside
// OK if process does all its persistent accesses in one thread
if ((os_boolean)getenv("OS_TOGGLE_MAPASIDE"))
{
    cout << "OS_TOGGLE_MAPASIDE optimization is ENABLED" << endl;
} else {
    cout << "OS_TOGGLE_MAPASIDE optimization is NOT enabled" << endl;
}

cout << buf << " end system parameters " << buf << buf <<
    "==" << endl;
//=====

// Compute structural info needed by the update operations,
// since these operations need to know which id"s should
// be used next.

int baseCnt = NumAssmPerAssm;
int complexCnt = 1;
for (int i = 1; i < NumAssmLevels-1; i++) {
    baseCnt = baseCnt * NumAssmPerAssm;
    complexCnt += complexCnt * NumAssmPerAssm;
}

```



```

nextBaseAssemblyId = TotalModules*baseCnt + 1;
nextComplexAssemblyId = TotalModules*complexCnt + 1;
nextAtomicId = TotalAtomicParts + 1;
nextCompositeId = TotalCompParts + 1;

int opIndex = 2;
int repeatCount = 1;
BenchmarkOp whichOp = Trav1;
int manyXACTS = 1;                                //set to 1 emw 8/95

// See if debug mode is desired, see which operation to run,
// and how many times to run it.                //already set to FALSE above emw 8/95

/* emw 8/95 commented out ***** */
// ParseCommandLine(argc, argv, opIndex, repeatCount, whichOp, manyXACTS);

// Now open the database, disable whole-segment transfers (in favor
// of cluster-level ones), and start up a transaction

char *dbname = getenv("DBNAME");
if (! dbname) {
    cout << "DBNAME environment variable NOT set" << endl;
//     dbname = "oo7li8";

/* emw 8/95 ***** */
// make sure dbname is set from the command line before each simulation

    return 1;
/* ***** */

} else {
    cout << "DBNAME environment variable IS set" << endl;
}
cout << "Using database " << dbname << endl;

oo7db = os_database::open(dbname);
if (oo7db == NULL) {
    fprintf(stderr, "ERROR: Cannot open OO7 Database.\n");
    exit(1);
}
cout << "database used is " << dbname << endl;

```

```

oo7db->set_read_whole_segment(0);
cout << "read_whole_segment mode is OFF for this"
    << " database" << endl;

// the set_fetch_policy is just experimental code
//oo7db->set_fetch_policy(os_fetch_segment,81920); // 20 pages
//cout << "set_fetch_policy is set to"
//    << " os_fetch_segment,81920)" << endl;

// Force schema validation to happen if it hasn't already
// happened. Also, set up fetch policy for the composite part segment

xact = os_transaction::begin();

AtomicPart_extent =
    (os_Set<AtomicPart*>*)
    (oo7db->find_root("AtomicPart_extent_root")->
     get_value());

os_segment *seg = os_segment::of(AtomicPart_extent);
seg->set_fetch_policy(os_fetch_page, 8192);
cout << "seg->set_fetch_policy(os_fetch_page,8192)," << endl;

//Opt code - qlb set once rather than in Query1b every time
static const os_coll_query & qlb =
    os_coll_query::create_pick("AtomicPart*",
    "(int)id == *(int*)ppartId",
    oo7db);

xact->commit();
delete xact;

// char* purgeVar;
// purgeVar = getenv("PURGE");
// if ((purgeVar != NULL) && (atoi(purgeVar) == 1)) {
//     do_transaction() {
//         printf("purging db\n");
//         Purge();
//         printf("done purging db\n");
//     }
// } else {
//     printf("not purging \n");
// }

```

```

// Actually run the darn thing

/* emw 8/95 new code to control number of operations ***** */
/* and convert input to the type the benchmark needs ***** */
    int n;
    int iter=0;
    char op[7];

    cout << "Enter the number of operations to simulate (1 - 500)" << endl;
    cin >> n;

    for (i=1;i<=n;i++) {
        //Get whichOp to run next--subset of all available operations
        inopns >> op;

        if (strcmp(op, "T1") == 0) {
            whichOp = Trav1;
        }
        else if (strcmp(op, "T2a") == 0) {
            whichOp = Trav2a;
        }
        else if (strcmp(op, "T3a") == 0) {
            whichOp = Trav3a;
        }
        else if (strcmp(op, "T6") == 0) {
            whichOp = Trav6;
        }
        else if (strcmp(op, "Q1") == 0) {
            whichOp = Query1;
        }
        else if (strcmp(op, "Q2") == 0) {
            whichOp = Query2;
        }
        else if (strcmp(op, "Q3") == 0) {
            whichOp = Query3;
        }
        else if (strcmp(op, "Q4") == 0) {
            whichOp = Query4;
        }
        else if (strcmp(op, "Q5") == 0) {
            whichOp = Query5;
        }
    }

```

```

else if (strcmp(op, "Q6") == 0) {
    whichOp = Query6;
}
else if (strcmp(op, "Q8") == 0) {
    whichOp = Query8;
}
else if (strcmp(op, "Insert") == 0) {
    whichOp = Insert;
}
else if (strcmp(op, "Delete") == 0) {
    whichOp = Delete;
}
else {
    cout << "error--unwanted operation" << endl;
    return 1;
}
/* emw 8/95 end of new code ***** */

/* emw 8/95 comment out old loop ***** */
/* but allow for reset of counters for each operation ***** */
//   for (int iter = 0; iter < repeatCount; iter++)
//   {
/* emw 8/95 end of commented out code ***** */

// reset counters here so that only the last iter is reported
if ((os_boolean)getenv("OS_COUNTERS_LT"))
{
    objectstore::reset_counters();
    cout << "OS_COUNTERS are RESET to zero at iteration "
        << iter << endl;
//} else {
// cout << "OS_COUNTERS are NOT reset to zero at iteration "
//     << iter << endl;
// }

////////////////////////////////////
//
// Run an OO7 Benchmark Operation
//
////////////////////////////////////

/* emw 8/95 alter output ***** */
//   printf("RUNNING OO7 BENCHMARK OPERATION %s, iteration = %d.\n",

```

```

//      argv[opIndex], iter);
cout << "Running Operation Number " << i << ", " << op << endl;
out << op;
/* emw 8/95 end of change ***** */

// get wall clock time
startWallTime = clock()/(CLOCKS_PER_SEC*1.0);

// Start a new transaction if either this is the first iteration
// of a multioperation transaction or we are running each
// operation as a separate transaction

if ((iter == 0) || (manyXACTS)) xact = os_transaction::begin();

// set random seed so "hot" runs are truly hot
srand(1);

AtomicPart_extent =
    (os_Set<AtomicPart*>*)
    (oo7db->find_root("AtomicPart_extent_root")->
     get_value());
CompositePart_extent =
    (os_Set<CompositePart*>*)
    (oo7db->find_root("CompositePart_extent_root")->
     get_value());
Document_extent =
    (os_Set<Document*>*)
    (oo7db->find_root("Document_extent_root")->
     get_value());
BaseAssembly_extent =
    (os_Set<BaseAssembly*>*)
    (oo7db->find_root("BaseAssembly_extent_root")->
     get_value());
Module_extent =
    (os_Set<Module*>*)
    (oo7db->find_root("Module_extent_root")->
     get_value());

const os_coll_query & q =
    os_coll_query::create_pick("Module*",
        "(int)id == (int)qmoduleId",
        oo7db);

```

```

// Use random module for the operation
int moduleId = (int) (rand () % TotalModules) + 1;
os_bound_query bq(q,os_keyword_arg("moduleId",moduleId) );

Module* module =
    Module_extent->query_pick(bq);

// Perform the requested operation on the chosen module

int count = 0;
RealWork = 0; // default is not to do work

switch (whichOp) {
    case Trav1:
        count = module->traverse(whichOp);
        sprintf(resultText, "Traversal 1 DFS visited %d atomic parts\n",
            count);
        break;

    case Trav1WW:
        RealWork = 1;
        whichOp = Trav1; // so traverse methods work correctly
        count = module->traverse(whichOp);
        whichOp = Trav1WW; // for next (hot) iteration
        sprintf(resultText, "Traversal 1 WW DFS visited %d atomic parts\n",
            count);
        break;

    case Trav2a:
        count = module->traverse(whichOp);
        sprintf(resultText, "Traversal 2A swapped %d pairs of (X,Y) coordinates\n",
            count);
        break;

    case Trav2b:
        count = module->traverse(whichOp);
        sprintf(resultText, "Traversal 2B swapped %d pairs of (X,Y) coordinates\n",
            count);
        break;

    case Trav2c:
        count = module->traverse(whichOp);
        sprintf(resultText, "Traversal 2C swapped %d pairs of (X,Y) coordinates\n",
            count);
        break;
}

```

```

case Trav3a:
    count = module->traverse(whichOp);
    sprintf(resultText, "Traversal 3A toggled %d dates.\n",
        count);

    break;
case Trav3b:
    count = module->traverse(whichOp);
    sprintf(resultText, "Traversal 3B toggled %d dates.\n",
        count);

    break;
case Trav3c:
    count = module->traverse(whichOp);
    sprintf(resultText, "Traversal 3C toggled %d dates.\n",
        count);

    break;
case Trav4:
    count = module->traverse(whichOp);
    sprintf(resultText, "Traversal 4: %d instances of the character found\n",
        count);

    break;
case Trav5do:
    count = module->traverse(whichOp);
    sprintf(resultText, "Traversal 5(DO): %d string replacements performed\n",
        count);

    break;
case Trav5undo:
    count = module->traverse(whichOp);
    sprintf(resultText, "Traversal 5(UNDO): %d string replacements
        performed\n",
        count);

    break;
case Trav6:
    count = module->traverse(whichOp);
    sprintf(resultText, "Traversal 6: visited %d atomic root parts.\n",
        count);

    break;
case Trav7:
    count = traverse7();
    sprintf(resultText, "Traversal 7: found %d assemblies using random atomic
        part.\n",
        count);

    break;
case Trav8:

```

```

count = module->scanManual();
sprintf(resultText, "Traversal 8: found %d instances of char in manual \n",
        count);
break;
case Trav9:
count = module->firstLast();
sprintf(resultText, "Traversal 9: match was %d \n",
        count);
break;
case Trav10:
// run traversal #1 on every module
count = 0;
whichOp = Trav1; // so object methods don't complain
for (moduleId = 1; moduleId <= TotalModules; moduleId++) {
    os_bound_query bq(q,os_keyword_arg("qmoduleId",moduleId));
    module =
        Module_extent->query_pick(bq);

    if (module == NULL) {
        printf("Could not find module %d \n", moduleId);
        os_transaction::abort();
        exit(1);
    }
    count += module->traverse(whichOp);
}
sprintf(resultText,
        "Traversal 10 visited %d atomic parts in %d modules \n",
        count, TotalModules);
whichOp = Trav10; // for next time around.
break;
case Query1:
count = query1();
sprintf(resultText, "Query one retrieved %d atomic parts \n",
        count);
break;
case Query1a:
count = query1a();
sprintf(resultText, "Query one a retrieved %d atomic parts \n",
        count);
break;
case Query1b:
count = query1b(q1b);
sprintf(resultText, "Query one b retrieved %d atomic parts \n",

```



```

        count);
    break;
case Query1WW:
    RealWork = 1;
    whichOp = Query1; // just in case...
    count = query1();
    whichOp = Query1WW; // for next (hot) iteration
    sprintf(resultText, "Query one WW retrieved %d atomic parts \n",
        count);
    break;
case Query2:
    count = query2();
    sprintf(resultText, "Query two retrieved %d qualifying atomic parts \n",
        count);
    break;
case Query3:
    count = query3();
    sprintf(resultText, "Query three retrieved %d qualifying atomic parts \n",
        count);
    break;
case Query4:
    count = query4();
    sprintf(resultText, "Query four retrieved %d (document, base assembly)
        pairs \n", count);
    break;
case Query4a:
    count = query4a();
    sprintf(resultText, "Query four a retrieved %d (document, base assembly)
        pairs \n", count);
    break;
case Query5:
    count = query5();
    sprintf(resultText, "Query five retrieved %d out-of-date base assemblies \n",
        count);
    break;
case Query6:
    count = query6();
    sprintf(resultText, "Query six retrieved %d out-of-date assemblies \n",
        count);
    break;
case Query7:
    count = query7();
    sprintf(resultText, "Query seven iterated through %d atomic parts \n",

```

```

        count);
    break;
case Query8:
    count = query8();
    sprintf(resultText, "Query eight found %d atomic part/document
        matches \n", count);
    break;
case Query8a:
    count = query8a();
    sprintf(resultText, "Query eight a found %d atomic part/document
        matches \n", count);
    break;
case Query8b:
    count = query8b();
    sprintf(resultText, "Query eight b found %d atomic part/document
        matches \n", count);
    break;
case Query8c:
    count = query8c();
    sprintf(resultText, "Query eight c found %d atomic part/document
        matches \n", count);
    break;
case Query8d:
    count = query8d();
    sprintf(resultText, "Query eight d found %d atomic part/document
        matches \n", count);
    break;
case Insert:
    insert1();
    sprintf(resultText, "Inserted %d composite parts (a total of %d atomic
        parts.)\n", NumNewCompParts,
        NumNewCompParts*NumAtomicPerComp);
    break;
case Delete:
    delete1();
    sprintf(resultText, "Deleted %d composite parts (a total of %d atomic
        parts )\n", NumNewCompParts,
        NumNewCompParts*NumAtomicPerComp);
    break;
case Reorg1:
    count = reorg1();
    sprintf(resultText, "Reorg1 replaced %d atomic parts \n", count);
    break;

```

```

case Reorg2:
    count = reorg2();
    sprintf(resultText, "Reorg2 replaced %d atomic parts.\n", count);
    break;
case WarmUpdate:
    // first do the t1 traversal to warm the cache
    count = module->traverse(Trav1);
    // then call T2 to do the update
    count = module->traverse(Trav2a);
    sprintf(resultText,
        "Warm update swapped %d pairs of (X,Y) coordinates.\n",
        count);
    break;
case MultiTrav1:
    count = module->traverse(whichOp);
    sprintf(resultText, "MultiTrav1 touched %d atomic parts.\n", count);
    break;
case MultiTrav2:
    count = module->traverse(whichOp);
    sprintf(resultText, "MultiTrav2 touched %d atomic parts.\n", count);
    break;
case MultiTrav3:
    count = module->traverse(whichOp);
    sprintf(resultText, "MultiTrav3 touched %d atomic parts.\n", count);
    break;
case MultiTrav4:
    count = module->traverse(whichOp);
    sprintf(resultText, "MultiTrav4 touched %d atomic parts.\n", count);
    break;
case MultiTrav5:
    count = module->traverse(whichOp);
    sprintf(resultText, "MultiTrav5 touched %d atomic parts.\n", count);
    break;
case MultiTrav6:
    count = module->traverse(whichOp);
    sprintf(resultText, "MultiTrav6 touched %d atomic parts.\n", count);
    break;
default:
    fprintf(stderr, "Sorry, that operation isn't available yet \n");
    os_transaction::abort();
    exit(1);
}
os_coll_query::destroy(q);

```

```

    if ((iter == repeatCount-1) || (manyXACTS)) {
        // end this transaction
        os_transaction::commit(xact);
    }

    // compute and report wall clock time
    endWallTime = clock()/(CLOCKS_PER_SEC*1.0);

    printf("OS, operation=%s, iteration=%d, elapsedTime=%f seconds\n",
//      argv[opIndex], iter,                // emw 8/95
        op, i,                            // emw 8/95
        (endWallTime - startWallTime));

    if (iter == 1) startWarmTime = startWallTime;

    // now print result string
    fprintf(stdout, resultText);

/* emw 8/95 output time to file ***** */
    out << "    " << (endWallTime - startWallTime) << "    ";
    out << count << endl;
/* emw 8/95 end of change ***** */

    if ((repeatCount > 2) && (iter == repeatCount-2))
    {
        printf("OS, operation=%s, average hot elapsedTime=%f seconds\n",
            argv[opIndex],
            (endWallTime - startWarmTime)/(repeatCount-2));
    }

    //////////////////////////////////////
    //
    // Shutdown
    //
    //////////////////////////////////////

    // print counters
    if ( (os_boolean)getenv("OS_COUNTERS") ||
        (os_boolean)getenv("OS_COUNTERS_LT") )
    {
        objectstore::print_counters();
        cout << "OS_COUNTERS printed" << endl;
    }

```

```

    } /* endfor */

    // close the database
    oo7db->close();

    // disable event hooks
    if ((os_boolean)getenv("OS_HOOKS")) {
        objectstore::disable_event_hooks();
        cout << "OS_HOOKS printed" << endl;
    }

    /* emw 8/95 close files ***** */
    inconfig.close();
    inopns.close();
    out.close();
    /* emw 8/95 end of new code ***** */

    // Exit
    exit(0);
}

```

## APPENDIX C

### MODIFYING DEGREE OF INHERITANCE IN OO7 CLASS DEFINITIONS

```
// OO7 oo7.h revised by E. M. Walk 8/95 for inheritance experiment
//   Revised class definitions to add two more levels of inheritance to the original OO7.h.
//Adds abstract classes RootObject and DesignObject
//   OO7.h was part of OO7 benchmark jointly developed by Carey, Dewitt, and
//Naughton at Univ. of Wisconsin and developers at Object Design, Inc., 1993 for
//ObjectStore Release 2.0.1.
//   OO7 was ported to ObjectStore 3.1 for OS/2 in 1995 by J. Brownsmith, IBM Santa
//Teresa Laboratory, and enhanced with optional operations and options for
//environmental variables. Because of changes this program is not comparable to the
//Univ. of Wisconsin version. The ported version was offered as-is, with no warranty
//as to correctness or quality. The use of parts of this benchmark for this dissertation are
//gratefully acknowledged.

#ifndef __oo7_h
#define __oo7_h

//-----
// Start with some necessary preliminaries
//-----

#define TypeSize      10
#define TitleSize     40
#define DummySize     1000

const int FALSE = 0;
const int TRUE = 1;

typedef enum {Complex, Base} AssemblyType;

typedef enum { Trav1, Trav1WW, Trav2a, Trav2b, Trav2c, Trav3a, Trav3b,
               Trav3c, Trav4, Trav5do, Trav5undo, Trav6, Trav7, Trav8,
               Trav9, Trav10, Query1, Query1a, Query1b,
               Query1WW, Query2, Query3, Query4, Query4a,
               Query5, Query5a, Query5b, Query5c, Query5d, Query5e, Query5f,
               Query6, Query7, Query8, Query8a, Query8b, Query8c,
               Query8d, Insert, Delete, Reorg1, Reorg2,
               WarmUpdate, MultiTrav1, MultiTrav2, MultiTrav3, MultiTrav4,
               MultiTrav5, MultiTrav6 } BenchmarkOp;
```

```

typedef enum {UpdateOne, UpdateAll, UpdateRepeat} UpdateType;
typedef enum {UpdateDirectionDo, UpdateDirectionUndo} UpdateDirectionType;

#include "PartIdSe h"

/*new abstract class emw 8/95 ***** */
class RootObject {
public:
    char                type[TypeSize],
    virtual int traverse(BenchmarkOp op) = 0;
};
/*new abstract class emw 8/95 ***** */

/*new abstract class emw 5/95 ***** */
class DesignObject : public RootObject {
public:
    os_backptr    bkpctr;
    //char        type[TypeSize];
    int           buildDate;
    int traverse(BenchmarkOp op) {return 0;};
};
/*new abstract class emw 5/95 ***** */

//-----
// AtomicPart objects are the primitives for building up designs
//   - modeled after the Sun/OO1 benchmark's parts
//-----

class Connection;
class CompositePart;

class AtomicPart : public RootObject {
public:
    os_backptr    bkpctr;
    os_indexable_member(AtomicPart,id,int) id;
    //char        type[TypeSize];
    os_indexable_member(AtomicPart,buildDate,int) buildDate;
    int           x, y;
    int           docId /* indexable */;
    os_relationship_m_1(AtomicPart,to,Connection,from,os_Bag<Connection*>) to;
    os_relationship_m_1(AtomicPart,from,Connection,to,os_Bag<Connection*>) from;
    os_relationship_l_1_m(AtomicPart,partOf,CompositePart,parts,CompositePart*) partOf;

```

```

AtomicPart(int ptId);
~AtomicPart();
void swapXY();
void toggleDate();
int traverse(BenchmarkOp op) {return 0;},           //emw 8/95
int traverse(BenchmarkOp op, PartIdSet& visitedIds);
void DoNothing();
};

```

```

// define global variable for AtomicPart class extent
extern os_Set<AtomicPart*>* AtomicPart_extent;

```

```

//-----
// Connection objects are used to wire AtomicParts together
//   - similarly, modeled after Sun/OO1 connections
//-----

```

```

class Connection : public RootObject {           //emw 8/95
public:
    //char          type[TypeSize];
    int            length;
    os_relationship_1_m(Connection,from,AtomicPart,to,AtomicPart*) from;
    os_relationship_1_m(Connection,to,AtomicPart,from,AtomicPart*) to;

    Connection(AtomicPart* fromPart, AtomicPart* toPart);
    int traverse(BenchmarkOp op) {return 0;};     //emw 8/95
};

```

```

//-----
// CompositeParts are parts constructed from AtomicParts
//   - entry in a library of reusable components
//   - implementation is a graph of atomic parts
//   - provides unit of significant access locality
//   - each has an associated (unique) document object
//-----

```

```

class Document;
class BaseAssembly;

```

```

/* inherit from DesignObject emw 5/95 ***** */
/* commented out inherited members ***** */

```



```

class CompositePart : public DesignObject {
public:
    // os_backptr    bkptr;
    os_indexable_member(CompositePart,id,int) id;
    // char          type[TypeSize];
    // int           buildDate;
    os_relationship_l_l(CompositePart,documentation,Document,part,Document*)
documentation;
    // index declared in CompositePart.C os_rel_l_l_body_options

os_relationship_m_m(CompositePart,usedInPriv,BaseAssembly,componentsPriv,os_Bag<
BaseAssembly*>) usedInPriv;
os_relationship_m_m(CompositePart,usedInShar,BaseAssembly,componentsShar,os_Bag
<BaseAssembly*>) usedInShar;
os_relationship_m_l(CompositePart,parts,AtomicPart,partOf,os_Set<AtomicPart*>)
parts;
    AtomicPart*      rootPart;

    CompositePart(int cpId);
    ~CompositePart();
    int traverse(BenchmarkOp op);
    int traverse7();
    int reorg1();
    int reorg2();
};
/* inherit from DesignObject emw 5/95 ***** */
/* ***** */

extern os_Set<CompositePart*>* CompositePart_extent;

//-----
// Document objects are used to hold a description of some particular
// CompositePart object
//-----

class Document : public RootObject {                                     //emw 8/95
public:
    os_backptr    bkptr;
    char          title[TitleSize]; /* indexable; <-see constructor */
    os_indexable_member(Document,id,int) id;
    char          *text;
    os_relationship_l_l(Document,part,CompositePart,documentation,CompositePart*)
part;

```

```

    Document(int cpId);
    ~Document();
    int searchText(char c);
    int replaceText(char *oldString, char* newString);
    void set_title(char this_title[]);
    int traverse(BenchmarkOp op) {return 0;};           //emw 8/95
};

```

```

extern os_Set<Document*>* Document_extent;

```

```

//-----
// Manual objects are used to hold a description of some particular
// module. Really just big documents, only associated with modules
// instead of CompositeParts
//-----

```

```

class Module;

```

```

class Manual : public RootObject {                     //emw 8/95
public:
    char          title[TitleSize];
    int           id;
    char          *text;
    int           textLen;
    os_relationship_1_1 (Manual,mod,Module,man,Module*) mod;

```

```

    Manual(int cpId);
    ~Manual();
    int searchText(char c);
    int replaceText(char *oldString, char* newString);
    int firstLast();
    int traverse(BenchmarkOp op){return 0;};           //emw 8/95
};

```

```

//-----
// Assembly objects are design instances built up recursively from
// from other Assembly objects and (at the leaves only) CompositeParts
//   - hierarchical (tree) structure for designs
//   - may share composite parts with other assemblies
//   - nonleaf and leaf assembly subtypes
//-----

```

```

class ComplexAssembly;

/* inherit from DesignObject emw 5/95 ***** */
/* commented out inherited members ***** */
class Assembly : public DesignObject {
public:
    // os_backptr    bkpPtr;
    os_indexable_member(Assembly,id,int) id;
    // char          type[TypeSize];
    // int           buildDate;

    os_relationship_l_m(Assembly,superAssembly,ComplexAssembly,subAssemblies,ComplexAssembly*) superAssembly;
    os_relationship_l_m(Assembly,module,Module,assemblies.Module*) module,

    Assembly(int);
    virtual int traverse(BenchmarkOp op);
    virtual int traverse7(PartIdSet& visitedComplexIds) = 0;
    virtual AssemblyType myType() = 0;
    void DoNothing();
};
/* inherit from DesignObject emw 5/95 ***** */
/* ***** */

class ComplexAssembly: public Assembly {
public:

    os_relationship_m_l(ComplexAssembly,subAssemblies,Assembly,superAssembly,os_Set<
    Assembly*>) subAssemblies;

    ComplexAssembly(int asId, ComplexAssembly* parentAssembly,
        int levelNo, Module* mod);
    int traverse(BenchmarkOp op);
    int traverse7(PartIdSet& visitedComplexIds);
    virtual AssemblyType myType() { return Complex; };
};

```

```

class BaseAssembly: public Assembly {
public:
    // os_backptr    bkpctr, // not needed - redundant

    os_relationship_m_m(BaseAssembly,componentsPriv,CompositePart,usedInPriv,os_Bag<
    CompositePart*>) componentsPriv;
    // index declared in Assembly.CPP os_rel_m_m_body_options

    os_relationship_m_m(BaseAssembly,componentsShar,CompositePart,usedInShar,os_Bag
    <CompositePart*>) componentsShar;

    BaseAssembly(int asId, ComplexAssembly* parentAssembly, Module* mod);
    ~BaseAssembly();
    int traverse(BenchmarkOp op);
    int traverse7(PartIdSet& visitedComplexIds);
    virtual AssemblyType myType() { return Base; };
};

extern os_Set<BaseAssembly*>* BaseAssembly_extent;

//-----
// Modules are the designs resulting from Assembly composition
//   - unit of scaleup for the benchmark database
//   - may share composite parts with other modules
//-----

/* inherit from DesignObject emw 5/95 ***** */
/* commented out inherited members ***** */
class Module : public DesignObject {
public:
    // os_backptr    bkpctr;
    os_indexable_member(Module,id,int) id;
    // char          type[TypeSize];
    // int           buildDate;
    os_relationship_l_l(Module,man,Manual,mod,Manual*) man;
    os_relationship_m_l(Module,assemblies,Assembly,module,os_Set<Assembly*>)
assemblies;
    ComplexAssembly*    designRoot;

    Module(int modId);
    ~Module();
    int traverse(BenchmarkOp op);
    int scanManual();

```

```

    int firstLast();
};
/* inherit from DesignObject emw 5/95 ***** */
/* ***** */

extern os_Set<Module*>* Module_extent;

//-----
// Dummy objects -- used for cache purging
//-----

class DummyObj {
public:
    char          dummy[DummySize];
};

#endif

```

## APPENDIX D

### MODIFYING DEGREE OF INHERITANCE IN OO7 TRAVERSAL OPERATIONS TO USE THREE LEVELS OF RUN-TIME POLYMORPHISM

```
//OO7 travers.cpp revised by E. M. Walk 8/95 (T1, T6 only) for inheritance experiment
// Executes traversals of Complex Assemblies, Base Assemblies, and Composite
//Parts using run-time polymorphism, inheritance from RootObject.
// Travers.cpp was part of OO7 benchmark jointly developed by Carey, Dewitt, and
//Naughton at Univ. of Wisconsin and developers at Object Design, Inc., 1993 for
//ObjectStore Release 2.0.1.
// OO7 was ported to ObjectStore 3.1 for OS/2 in 1995 by J. Brownsmith, IBM Santa
//Teresa Laboratory, and enhanced with optional operations and options for
//environmental variables. Because of changes this program is not comparable to the
//Univ. of Wisconsin version. The ported version was offered as-is, with no warranty
//as to correctness or quality. The use of parts of this benchmark for this dissertation are
//gratefully acknowledged.
```

```
#include <string.h>
#include <stdio.h>
//the #include <task.h> contains delay fn on OS/2
#include <task.h>
//the following #include <stdlib.h> is for the rand() fn
#include <stdlib.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include <ostore/relat.hh>
#include "OO7.h"
#include "BenchPar.h"
#include "GenParam.h"
#include "VarParam.h"

////////////////////////////////////
//
// Traverse - DFS traverse module. Upon reaching a base assembly,
// visit all referenced composite parts. At each composite part,
// take an action that depends on which traversal variant has been
// requested.
//
////////////////////////////////////

extern os_database* oo7db;
extern int debugMode;
```

```
extern void PrintOp(BenchmarkOp op);
```

```
////////////////////////////////////
//
// Module Method for Traversal
//
////////////////////////////////////
```

```
int Module::traverse(BenchmarkOp op)
```

```
{
    if (debugMode) {
        printf("Module::traverse(id = %d, op = ", id);
        PrintOp(op);
        printf(")\n");
    }
}
```

```
// now traverse the assembly hierarchy
```

```
RootObject* rol;
```

```
//emw 8/95
```

```
rol = designRoot;
```

```
return rol->traverse(op);
```

```
//ptr to Complex Assembly emw 8/95
```

```
}
```

```
/* ***** */
```

```
/* Assembly Method for Traversal (no longer pure virtual) emw 8/95 */
```

```
int Assembly::traverse(BenchmarkOp op)
```

```
{
    printf("OOPS----This should not execute!\n");
    return 0;
}
/* ***** */
```

```
////////////////////////////////////
//
// ComplexAssembly Method for Traversal
//
////////////////////////////////////
```

```
int ComplexAssembly::traverse(BenchmarkOp op)
```

```

{
    if (debugMode) {
        printf("\tComplexAssembly::traverse(id = %d, op = ", id);
        PrintOp(op);
        printf(")\n");
    }

    // prepare for and execute the traversal
    int count = 0;
    Assembly* assm;
    RootObject* ro2; //emw 8/95
    ro2 = assm; //pointer to Assembly emw 8/95
    os_cursor cursl(subAssemblies);

    if (op < MultiTravl) {

        // fully traverse the assembly hierarchy
        for (ro2 = (Assembly*)cursl.first();
            cursl.more(); ro2 = (Assembly*)cursl.next()) {
            count += ro2->traverse(op); //emw 8/95
        }

    } else { //not changed (not used)

        // randomly walk down the assembly hierarchy
        int curPath = 0;
        int randPath = (int) (rand () % NumAssmPerAssm);
        for (assm = (Assembly*)cursl.first();
            cursl.more(); assm=(Assembly*)cursl.next()) {
            if (curPath++ == randPath) {
                count += assm->traverse(op);
                break;
            }
        }
    }

    // lastly, return the result
    return count;
}

```



```

////////////////////////////////////
//
// BaseAssembly Method for Traversal
//
////////////////////////////////////

int BaseAssembly::traverse(BenchmarkOp op)
{
    if (debugMode) {
        printf("\t\tBaseAssembly::traverse(id = %d, op = ", id);
        PrintOp(op);
        printf(")\n");
    }

    // prepare for and execute the traversal
    int count = 0;
    CompositePart* comp;
    RootObject* ro3; //emw 8/95
    ro3 = comp; //pointer to Compos. Part emw 8/95

    os_cursor curs2(componentsPriv);

    if (op < MultiTrav1) {

        // fully traverse each of the assembly's private components
        for (ro3 = (CompositePart*)curs2.first();
            curs2.more(); ro3=(CompositePart*)curs2.next()) {
            count += ro3->traverse(op); //emw 8/95
        }

    } else { //not changed (not used)

        // first, traverse private and/or shared composite parts

        if (op < MultiTrav3 || op > MultiTrav4) {
            for (comp = (CompositePart*)curs2.first();
                curs2.more(); comp=(CompositePart*)curs2.next()) {
                count += comp->traverse(Trav1);
            }
        }

        if (op > MultiTrav2) {

```

```

        os_cursor curs3(componentsShar);
        for (comp = (CompositePart*)curs3.first();
             curs3.more(); comp=(CompositePart*)curs3.next()) {
            count += comp->traverse(Trav1);
        }
    }

    // next, sleep for awhile to reduce server disk contention
    //sched::delay((long)MultiSleepTime);

    // also, perform an update traversal if one is desired

    if (op == MultiTrav2 || op == MultiTrav5 || op == MultiTrav6) {
        os_cursor curs4(componentsPriv);
        for (comp = (CompositePart*)curs4.first();
             curs4.more(); comp=(CompositePart*)curs4.next()) {
            count += comp->traverse(Trav2b);
        }
    } else if (op == MultiTrav4) {
        os_cursor curs5(componentsShar);
        for (comp = (CompositePart*)curs5.first();
             curs5.more(); comp=(CompositePart*)curs5.next()) {
            count += comp->traverse(Trav2b);
        }
    }

}

// lastly, return the result

return count;
}

////////////////////////////////////
//
// CompositePart Method for Traversal
//
////////////////////////////////////

```

```

int CompositePart::traverse(BenchmarkOp op)
{
    if (debugMode) {
        printf("\t\t\tCompositePart::traverse(id = %d, op = ", id);
        PrintOp(op);
        printf("\n");
    }

    if ((op >= Trav1) && (op <= Trav3c)) {

        // do parameterized DFS of atomic part graph
        PartIdSet visitedIds;
        return rootPart->traverse(op, visitedIds);           //pointer to AtomicPart
                                                             //does not inherit this from RootObject emw 8/95
    } else if (op == Trav4) {

        // search document text for a certain character
        return documentation->searchText('I');

    } else if (op == Trav5do) {

        // conditionally change initial part of document text
        return documentation->replaceText("I am", "This is");

    } else if (op == Trav5undo) {

        // conditionally change back initial part of document text
        return documentation->replaceText("This is", "I am");

    } else if (op == Trav6) {

        // visit the root part only (it knows how to handle this)
        PartIdSet visitedIds;
        return rootPart->traverse(op, visitedIds);

    } else {

        // composite part's dont respond to other traversals
        printf("*** CompositePart::PANIC -- illegal traversal!!! ***\n");
    }
}

```

```

////////////////////////////////////
//
// AtomicPart Method for Traversal
//
////////////////////////////////////

int AtomicPart::traverse(BenchmarkOp op, PartIdSet& visitedIds)
{
    if (debugMode) {
        printf("\t\t\t\tAtomicPart::traverse(id = %d, op = ", id);
        PrintOp(op);
        printf(")\n");
    }

    int i;
    int count = 0;

    switch (op) {

    case Trav1:

        // just examine the part
        count += 1;
        DoNothing();
        break;

    case Trav2a:

        // swap X and Y if first part
        if (visitedIds.empty()) {
            swapXY();
            count += 1;
        }
        break;

    case Trav2b:

        // swap X and Y
        swapXY();
        count += 1;
        break;
    }
}

```

case Trav2c:

```
// swap X and Y repeatedly
for (i = 0; i < UpdateRepeatCnt, i++) {
    swapXY(),
    count += 1;
}
break;
```

case Trav3a:

```
// toggle date if first part
if (visitedIds.empty()) {
    toggleDate();
    count += 1;
}
break;
```

case Trav3b:

```
// toggle date
toggleDate();
count += 1;
break;
```

case Trav3c:

```
// toggle date repeatedly
for (i = 0; i < UpdateRepeatCnt, i++) {
    toggleDate();
    count += 1;
}
break;
```

case Trav6:

```
// examine only the root part
count += 1;
DoNothing();
return count;
```

default:

```

        // atomic parts don't respond to other traversals
        printf("*** AtomicPart::PANIC -- illegal traversal!!! ***\n");

    }

    // now, record the fact that we've visited this part
    visitedIds.insert(id);

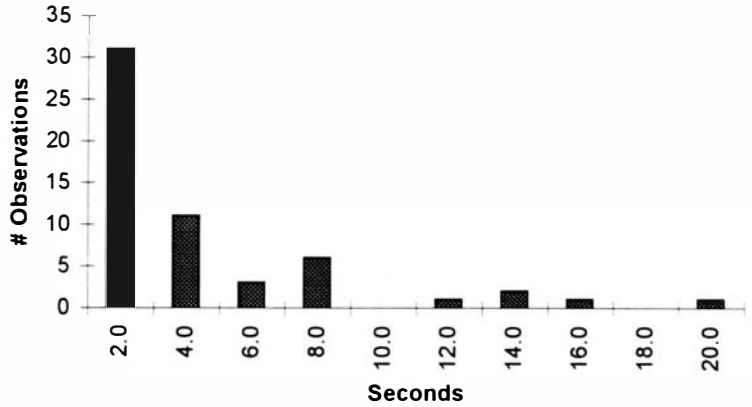
    // finally, continue with DFS of atomic parts graph

    os_cursor curs6(to);
    for (Connection* conn = (Connection*)curs6.first();
         curs6.more(); conn=(Connection*)curs6.next()) {
        if (!visitedIds.contains(conn->to->id)) {
            count += conn->to->traverse(op, visitedIds);
        }
    }

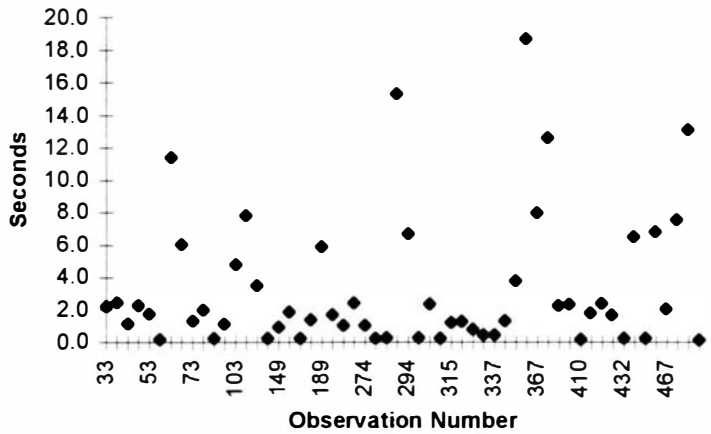
    return count;
}

```

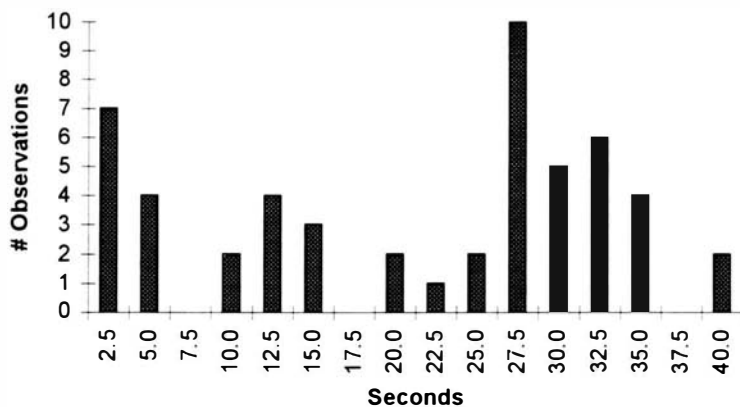
**Appendix E.1**  
**Frequency Distribution for Q1 Response Times,**  
**Database 7/200/3/3**



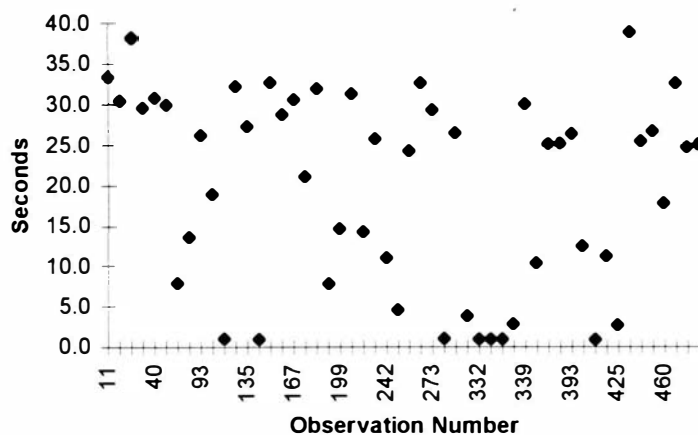
**Appendix E.2**  
**Q1 Response Times, In Order of Occurrence in**  
**Workload, Database 7/200/3/3**



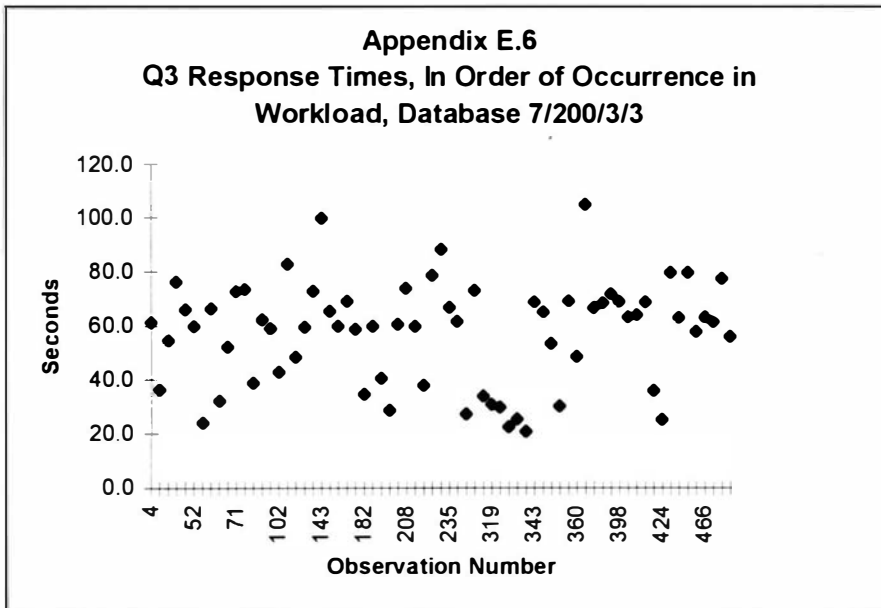
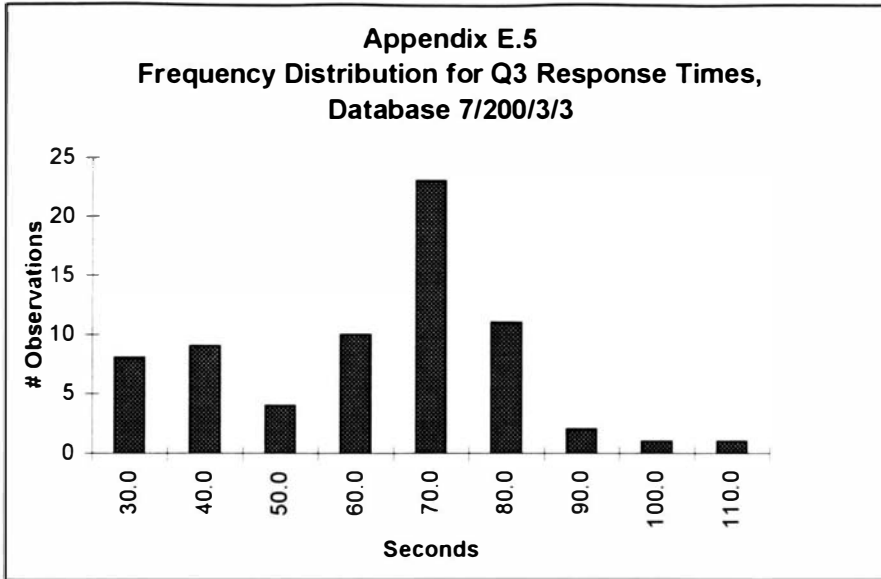
**Appendix E.3**  
**Frequency Distribution for Q2 Response Times,**  
**Database 7/200/3/3**



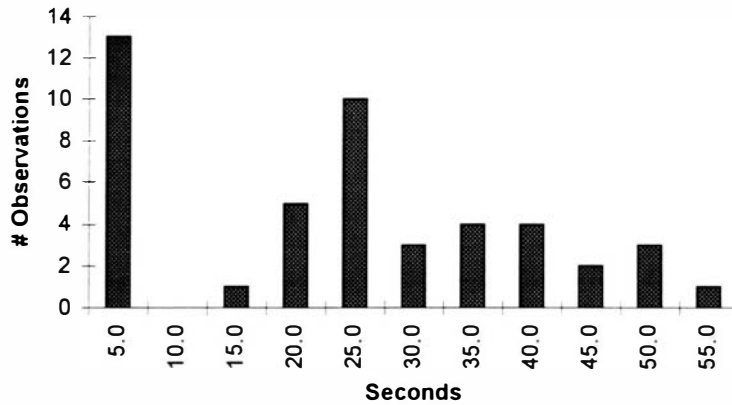
**Appendix E.4**  
**Q2 Response Times, In Order of Occurrence in**  
**Workload, Database 7/200/3/3**



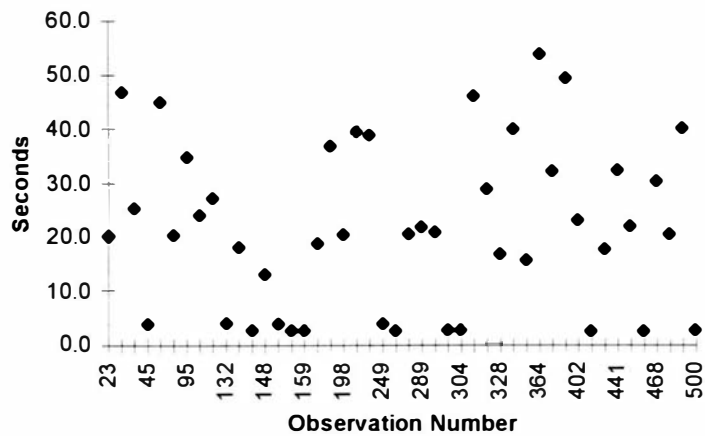




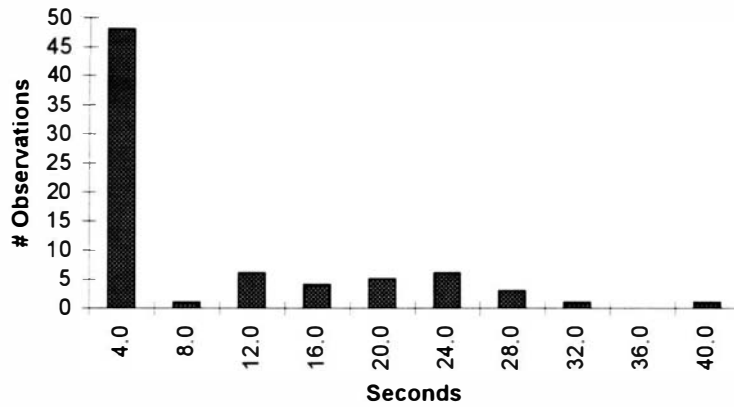
**Appendix E.7**  
**Frequency Distribution for Q4 Response Times,**  
**Database 7/200/3/3**



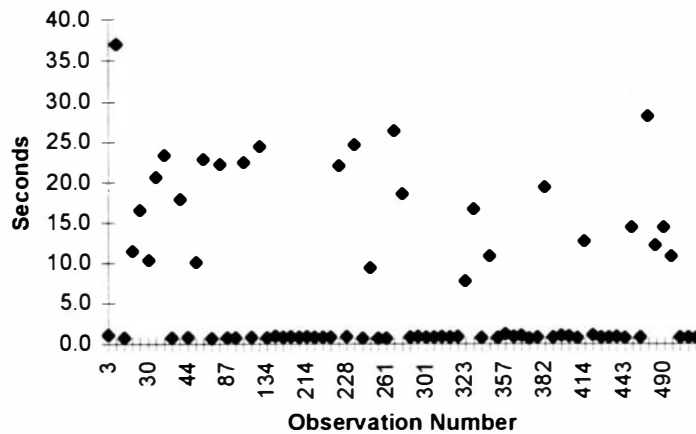
**Appendix E.8**  
**Q4 Response Times, In Order of Occurrence in**  
**Workload, Database 7/200/3/3**



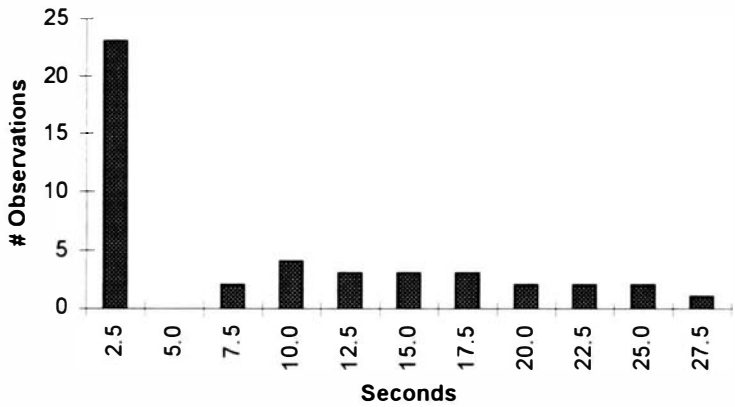
**Appendix E.9**  
**Frequency Distribution for Q5 Response Times,**  
**Database 7/200/3/3**



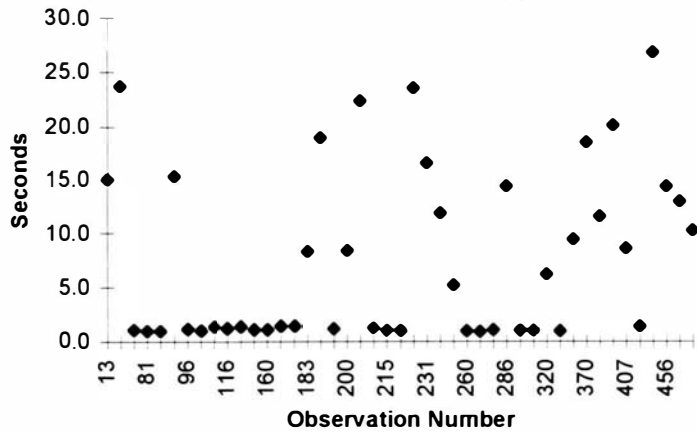
**Appendix E.10**  
**Q5 Response Times, In Order of Occurrence in**  
**Workload, Database 7/200/3/3**



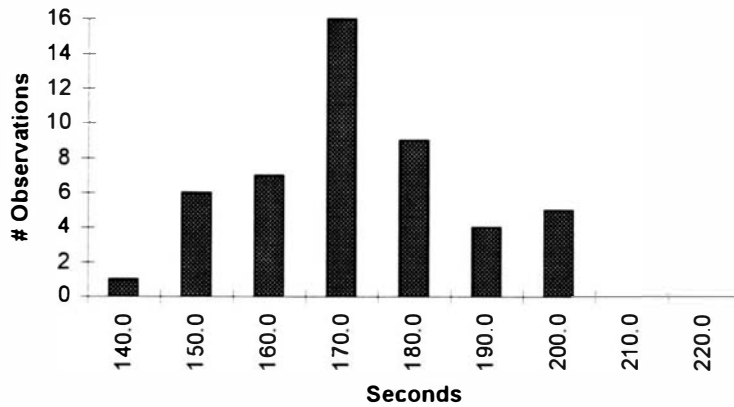
**Appendix E.11**  
**Frequency Distribution for Q6 Response Times,**  
**Database 7/200/3/3**



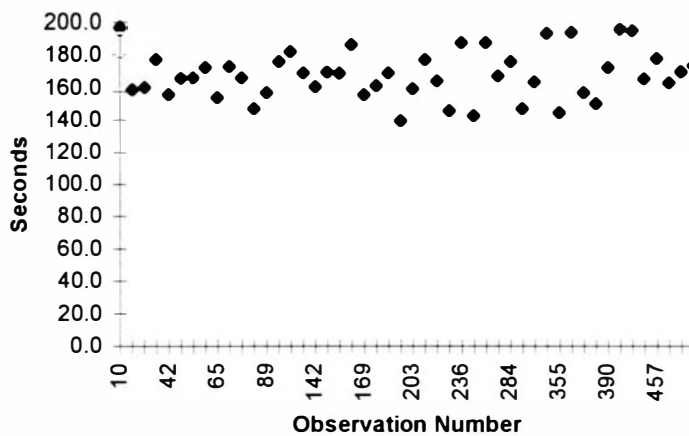
**Appendix E.12**  
**Q6 Response Times, In Order of Occurrence in**  
**Workload, Database 7/200/3/3**



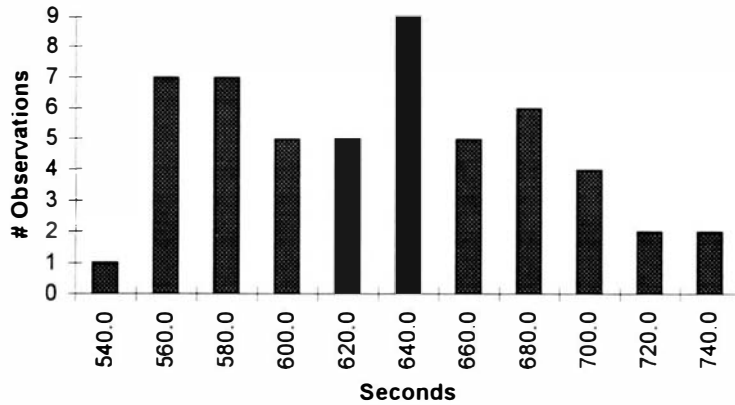
**Appendix E.13**  
**Frequency Distribution for Q8 Response Times,**  
**Database 7/200/3/3**



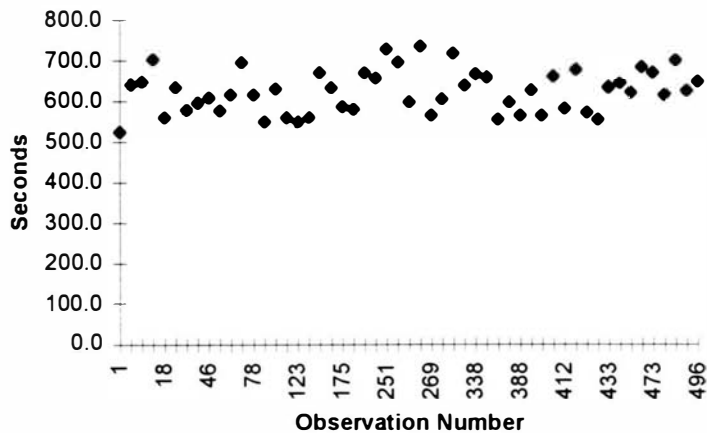
**Appendix E.14**  
**Q8 Response Times, In Order of Occurrence in**  
**Workload, Database 7/200/3/3**



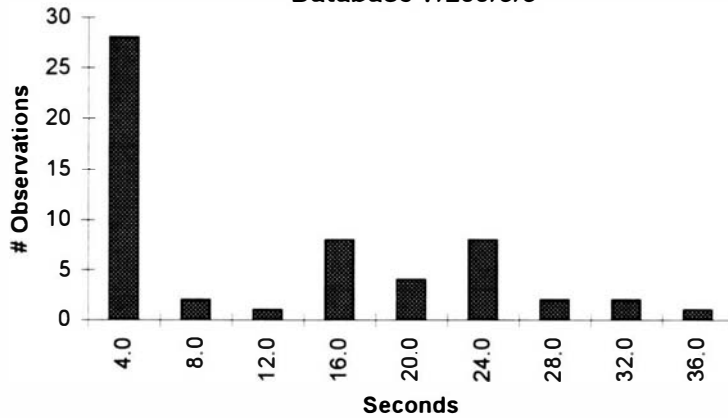
**Appendix E.15**  
**Frequency Distribution for T1 Response Times,**  
**Database 7/200/3/3**



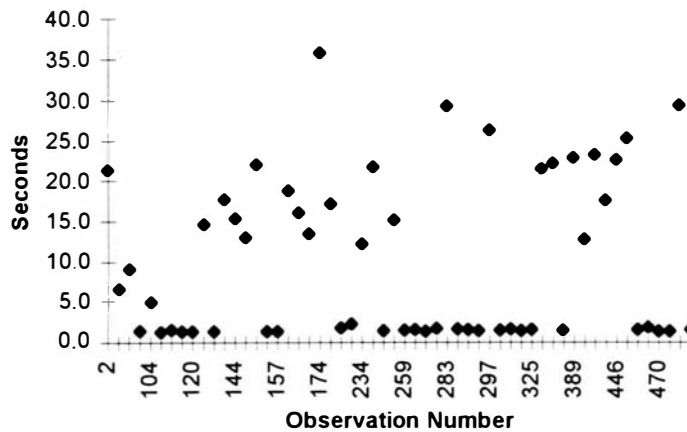
**Appendix E.16**  
**T1 Response Times, In Order of Occurrence in**  
**Workload, Database 7/200/3/3**

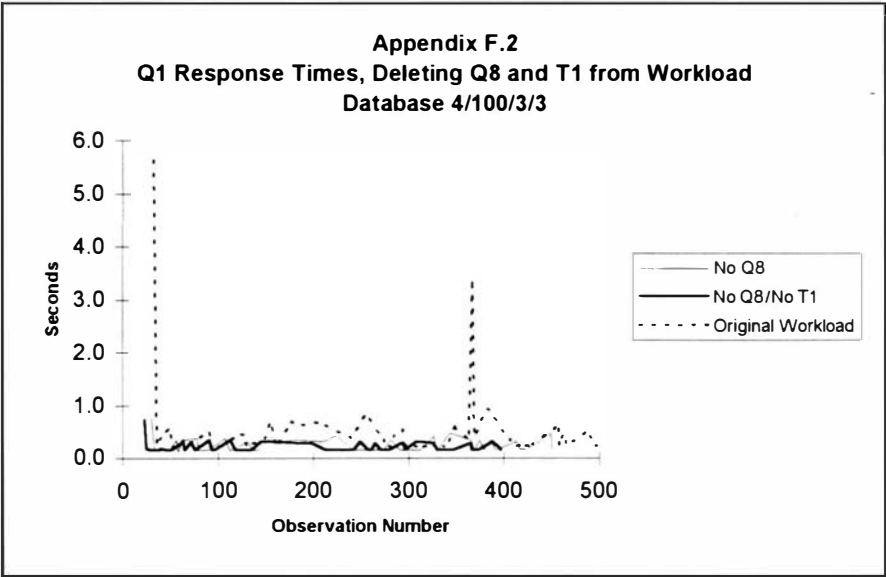
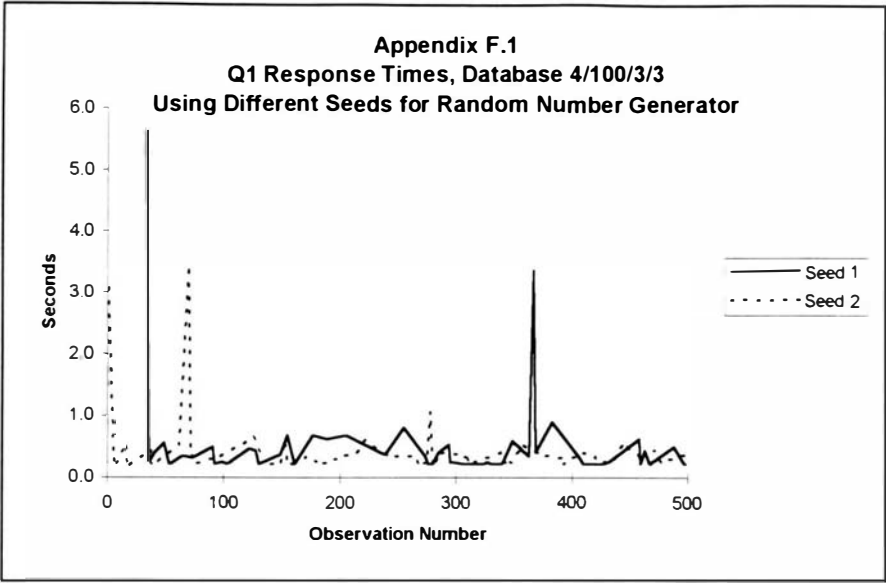


**Appendix E.17**  
**Frequency Distribution for T6 Response Times,**  
**Database 7/200/3/3**

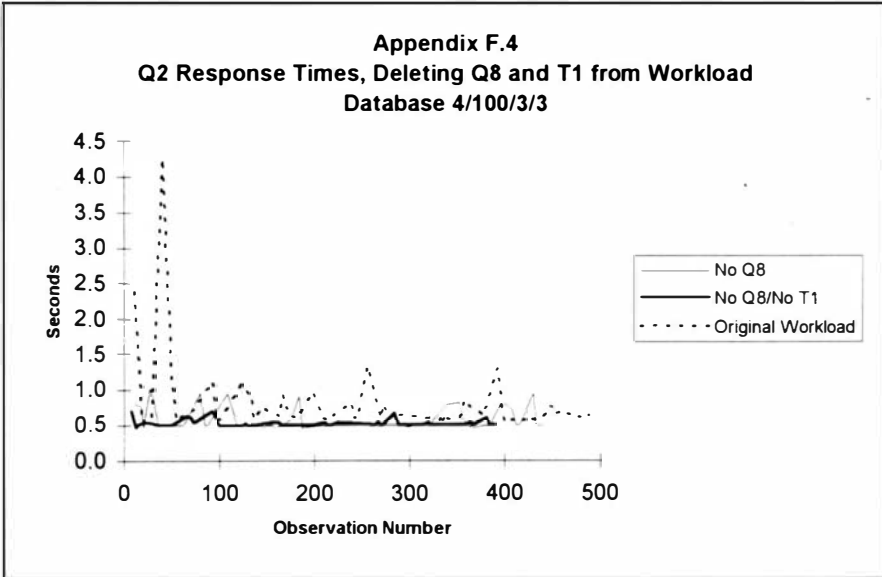
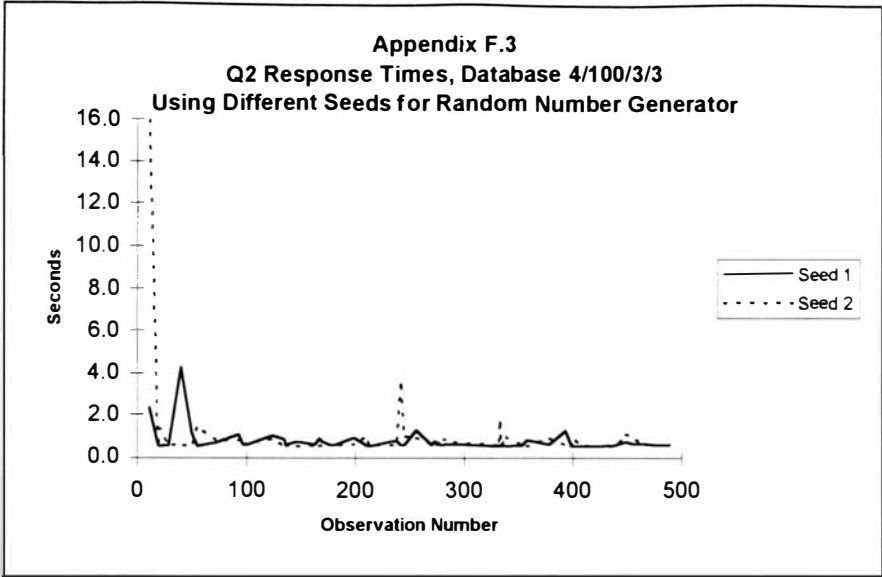


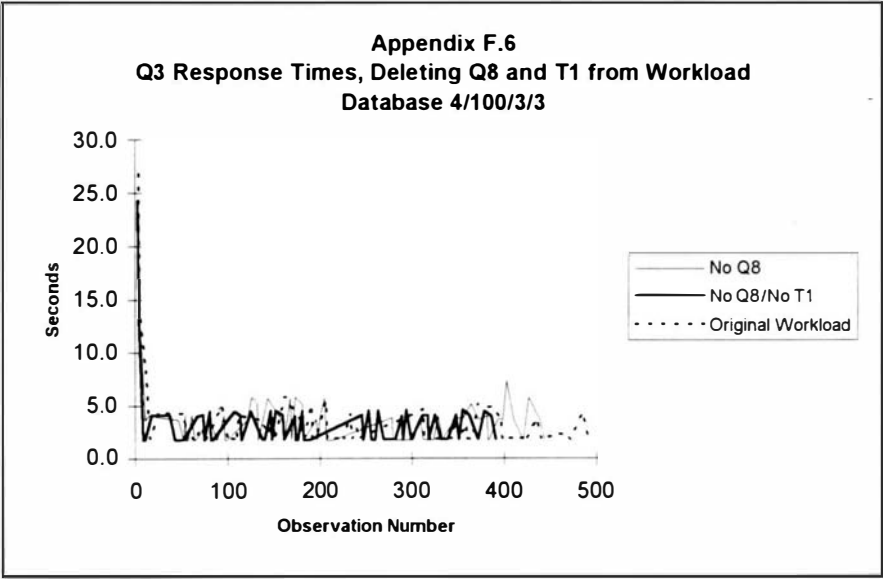
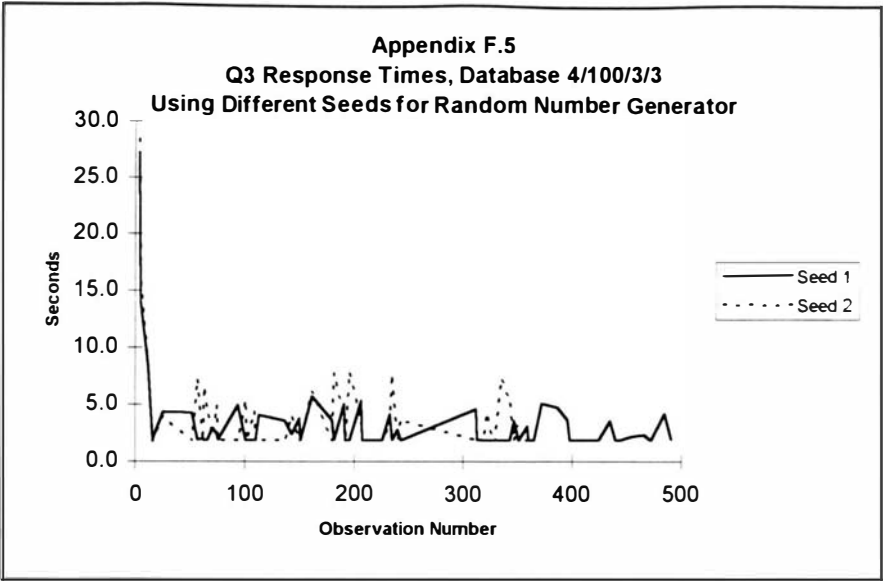
**Appendix E.18**  
**T6 Response Times, In Order of Occurrence in**  
**Workload, Database 7/200/3/3**

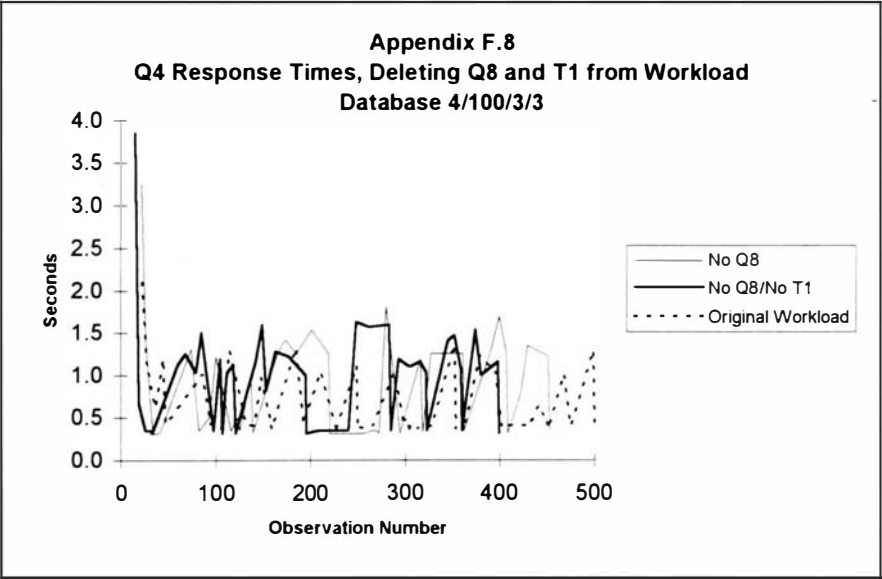
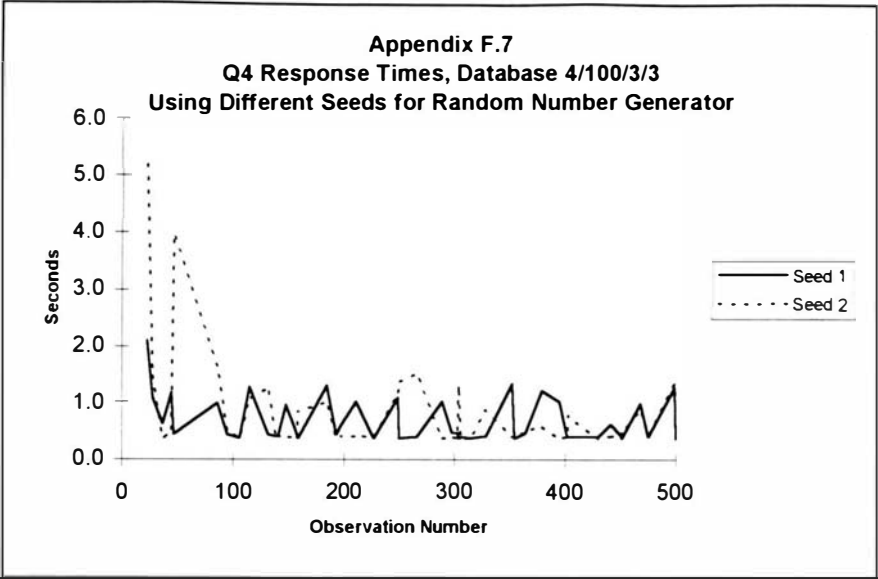


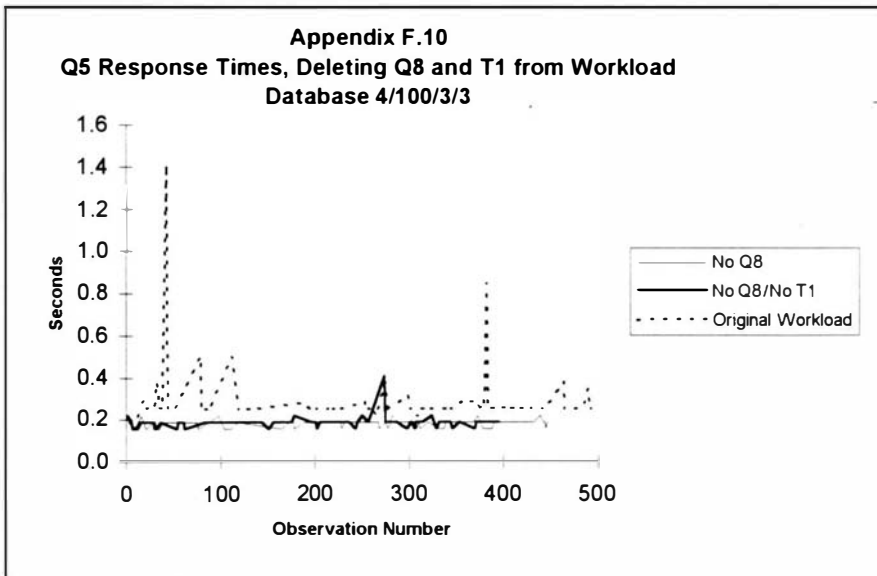
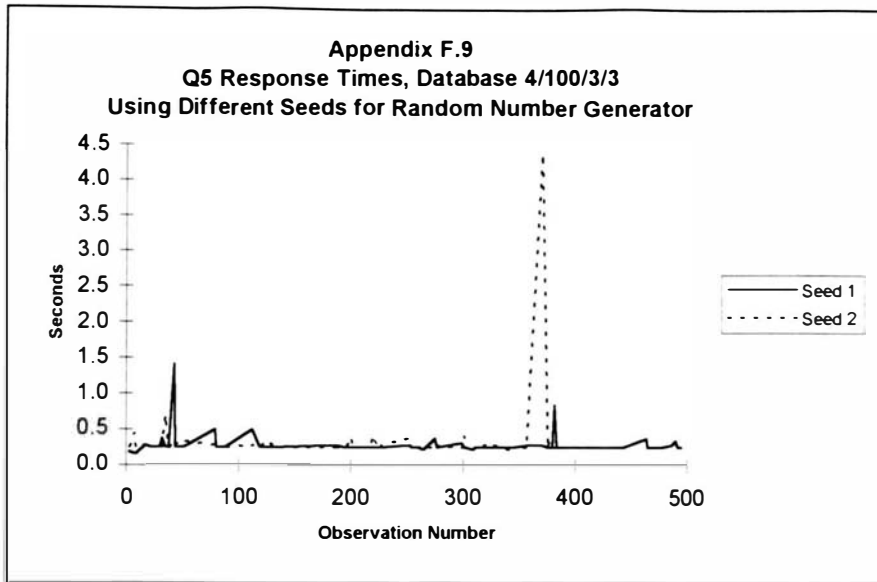


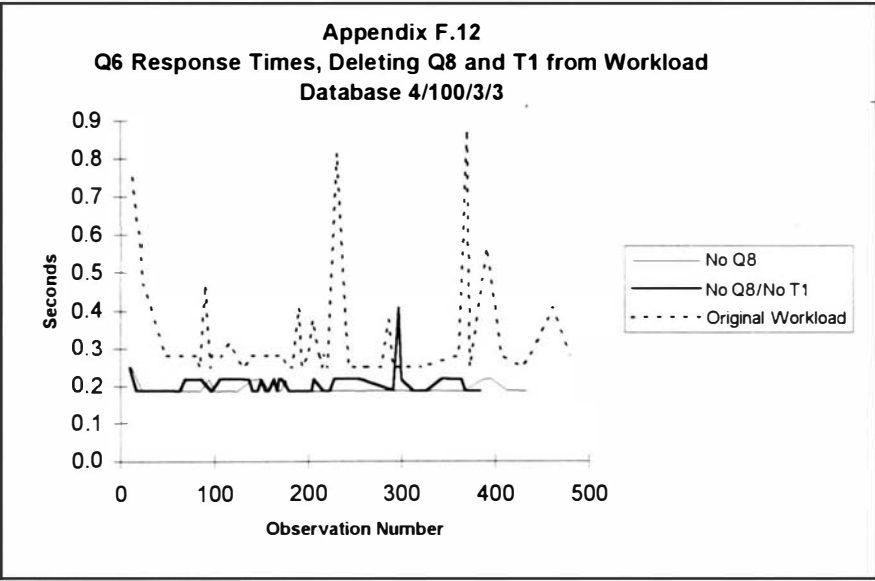
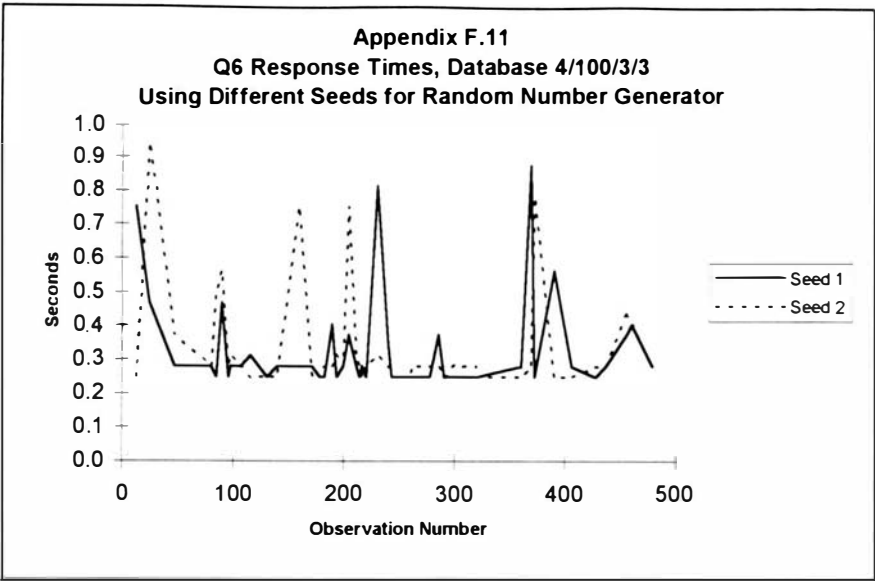


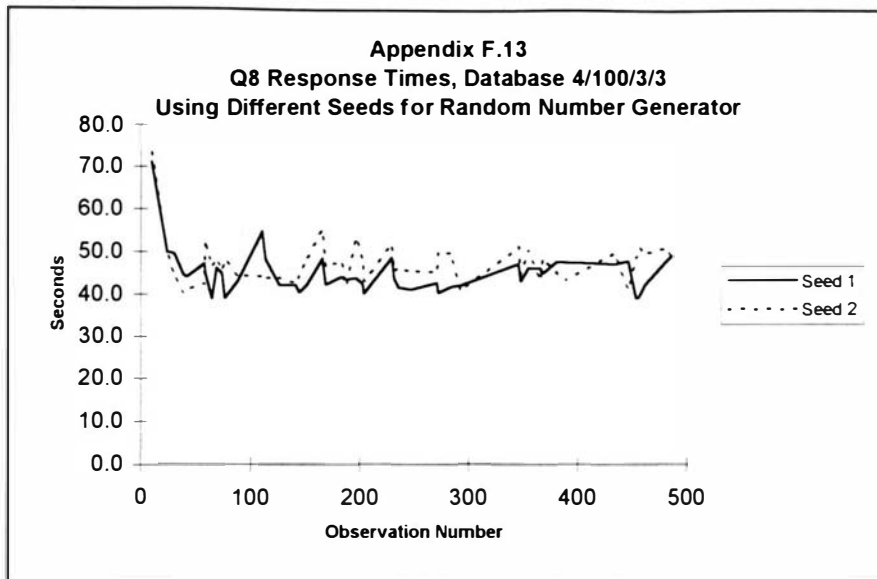


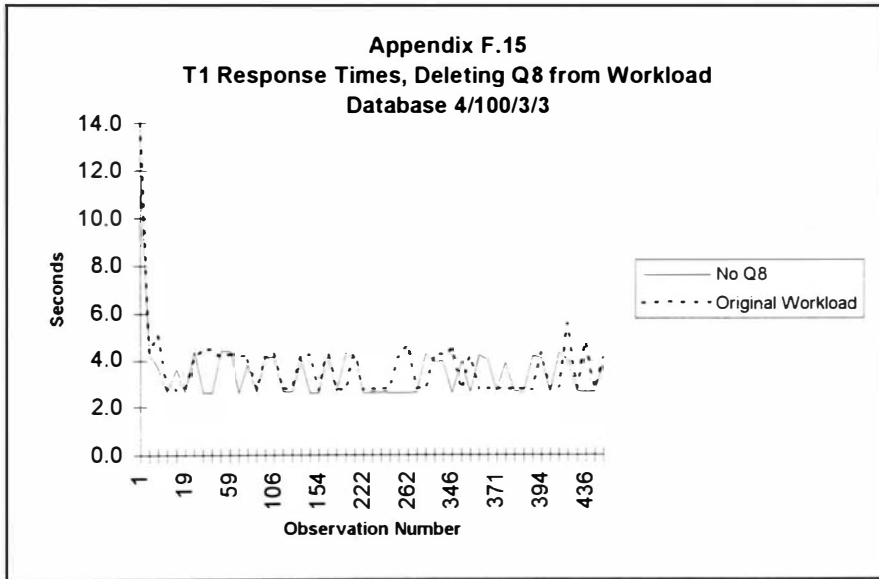
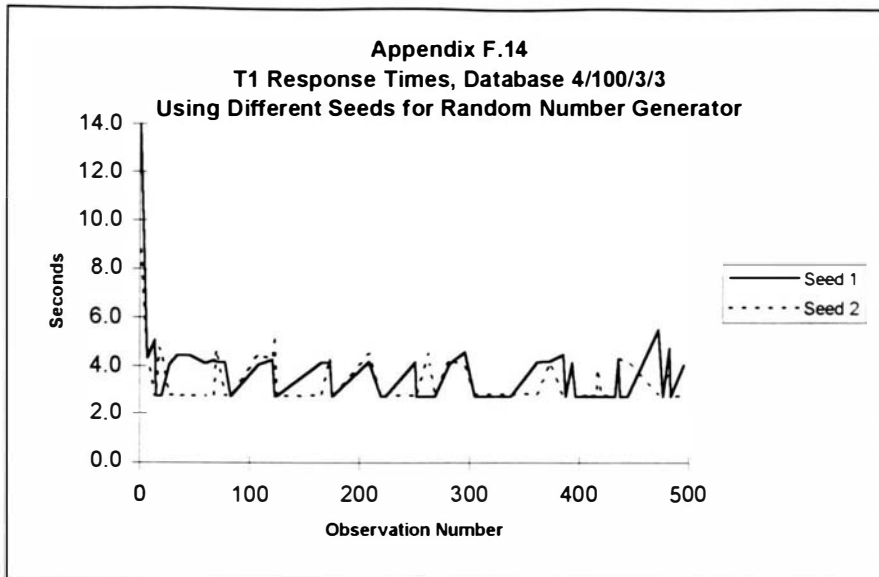


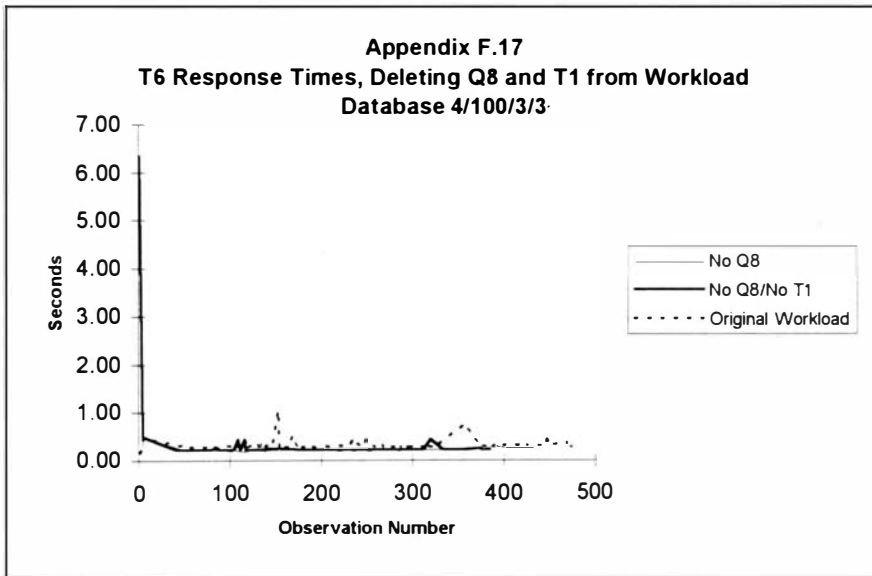
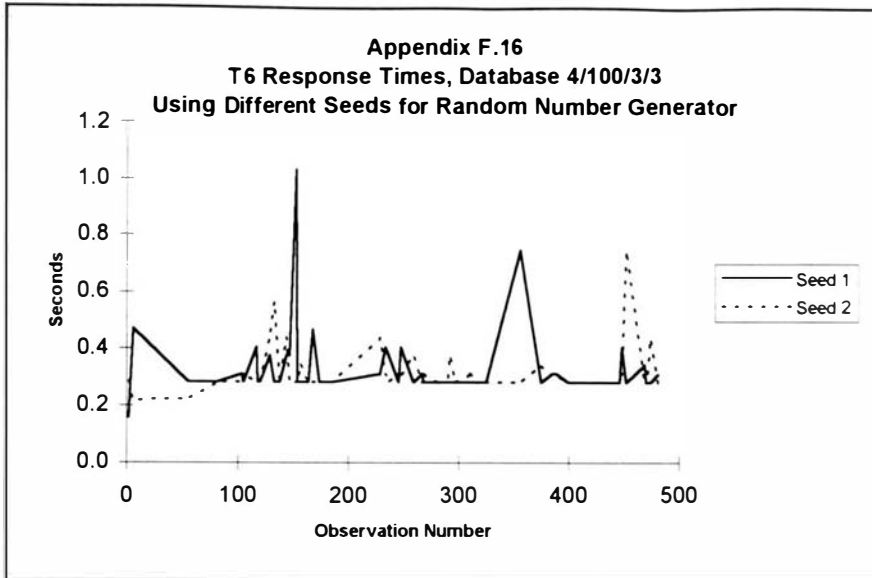














## Vita

