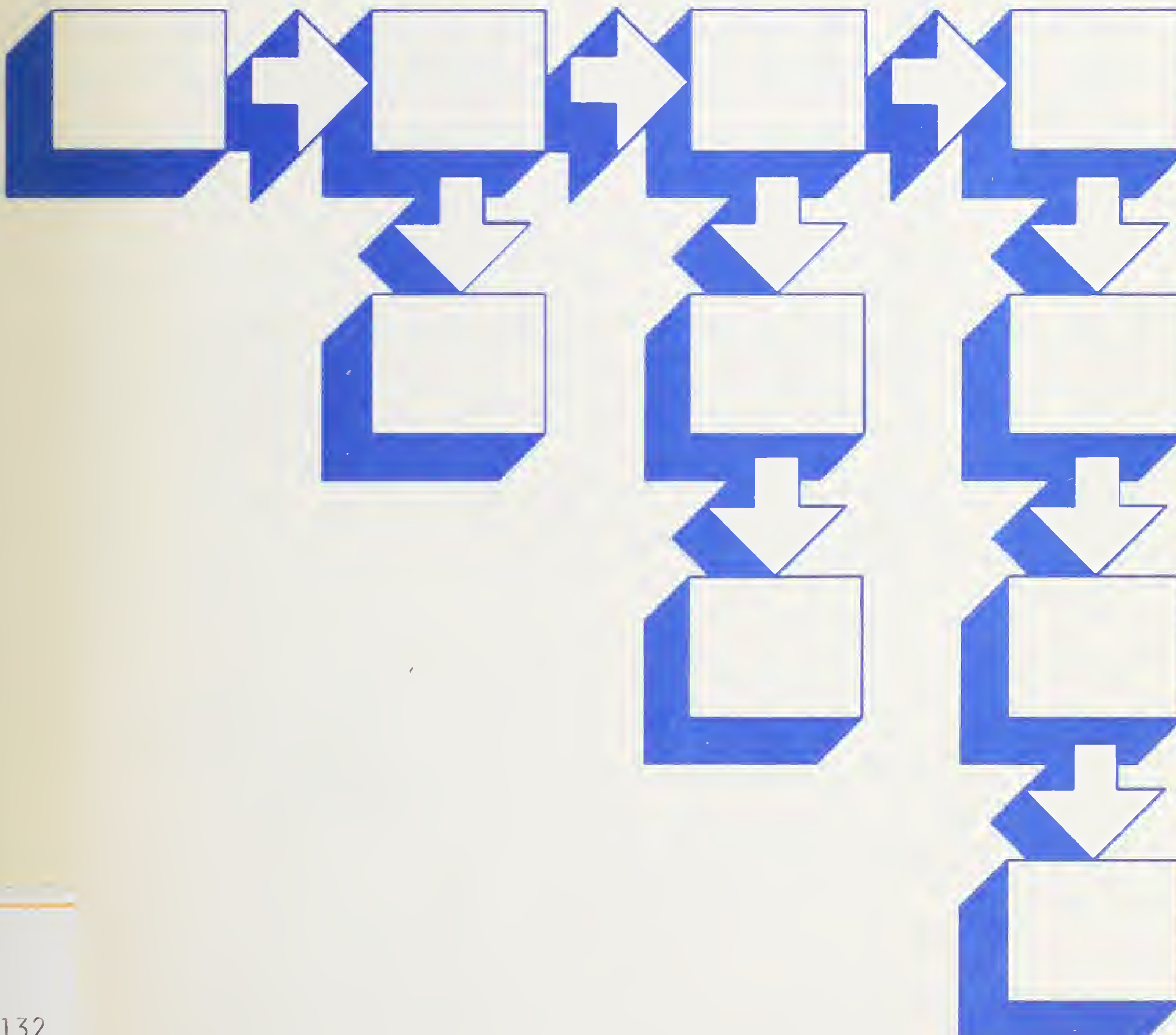
National Bureau
of Standards

Computer Science and Technology

NBS Special Publication 500-132

Benchmark Analysis of Database Architectures: A Case Study

Daniel R. Benigni, Editor



The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, the Institute for Computer Sciences and Technology, and the Institute for Materials Science and Engineering.

The National Measurement Laboratory

Provides the national system of physical and chemical measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; provides advisory and research services to other Government agencies; conducts physical and chemical research; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

- Basic Standards[2]
- Radiation Research
- Chemical Physics
- Analytical Chemistry

The National Engineering Laboratory

Provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

- Applied Mathematics
- Electronics and Electrical Engineering[2]
- Manufacturing Engineering
- Building Technology
- Fire Research
- Chemical Engineering[2]

The Institute for Computer Sciences and Technology

Conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following centers:

- Programming Science and Technology
- Computer Systems Engineering

The Institute for Materials Science and Engineering

Conducts research and provides measurements, data, standards, reference materials, quantitative understanding and other technical information fundamental to the processing, structure, properties and performance of materials; addresses the scientific basis for new advanced materials technologies; plans research around cross-country scientific themes such as nondestructive evaluation and phase diagram development; oversees Bureau-wide technical programs in nuclear reactor radiation research and nondestructive evaluation; and broadly disseminates generic technical information resulting from its programs. The Institute consists of the following Divisions:

- Inorganic Materials
- Fracture and Deformation[3]
- Polymers
- Metallurgy
- Reactor Radiation

---

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Gaithersburg, MD 20899.
[2]Some divisions within the center are located at Boulder, CO 80303.
[3]Located at Boulder, CO, with some elements at Gaithersburg, MD.

Computer Science and Technology

NBS Special Publication 500-132

Benchmark Analysis of Database Architectures: A Case Study

Daniel R. Benigni, Editor

Center for Programming Science and Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Gaithersburg, MD 20899

Prepared by:

S. Bing Yao
Alan R. Hevner

Software Systems Technology, Inc.
College Park, MD 20740

Issued October 1985

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

Library of Congress Catalog Card Number: 85-600599
National Bureau of Standards Special Publication 500-132
Natl. Bur. Stand. (U.S.), Spec. Publ. 500-132, 198 pages (Oct. 1985)
CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1985

For sale by the Suoerintendent of Documents, U.S. Government Printing Office, Washington, DC 20402

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES
(continued)

LIST OF FIGURES
(continued)

LIST OF TABLES

BENCHMARK ANALYSIS OF DATABASE ARCHITECTURES: A CASE STUDY

Daniel R. Benigni, Editor

This report presents an application of the generalized performance analysis methodology for the benchmarking of database systems reported in NBS Special Publication 500-118 [BENI 84]. The principal objectives of this report are to benchmark the performance of three distinct database system architectures: a microcomputer database system, a minicomputer database system, and a database machine. This report not only proves the viability of the benchmarking methodology in evaluating real systems, but it also provides comparable observations as to the capabilities of database systems based upon different architectures. Together with NBS Special Publication 500-118, this report serves as a reference for the benchmarking of database systems by providing a complete description of the benchmarking framework and a detailed application showing how to implement it.

Keywords: Benchmark execution; benchmark methodology; benchmark workload; database systems; DBMS; indexing; performance evaluation; query complexity; response time.

FOREWORD

This report is one of a continuing series of NBS publications in the area of data management technology. It concentrates on actual benchmark experiments with three database system architectures using the benchmarking methodology developed in NBS Special Publication 500-118 [BENI 84].

Other NBS publications addressing various aspects of data management system selection include: FIPS 77 [NBS 80], NBS Special Publication 500-108 [GALL 84], and FIPS 110 [NBS 85].

In this report the methodology is summarized and the results of the three benchmark experiments are presented. The final report is structured as follows:

Section 1 outlines the background and objectives of the project.

Section 2 provides a summary of the benchmark methodology developed for this project.

Sections 3-5 present the benchmark parameters that remain constant over the three benchmark experiments. These parameters are the test data, the benchmark workload, and the performance measurement and evaluation criteria.

Sections 6-8 present the details of the benchmark experiments for each of the three systems.

Section 9 completes the report by discussing the observations drawn from analyzing the benchmark results over the three database system architectures.

The appendices provide complete details of the original data tape (Appendix A), the benchmark workload queries (Appendix B), the performance data from the benchmarks (Appendix C), and the selection, implementation schedule, and benchmark execution for each tested system (Appendix D).

DISCLAIMER

This report identifies database management systems by trade names when necessary to provide a descriptive characterization of their features. Inclusion of a system in this report in no way implies a recommendation or endorsement by the National Bureau of Standards, and the presentation should not be construed as a certification that any system provides the indicated capabilities. Similarly the omission of a system does not imply that its capabilities are less than those of the included systems. The data presented in this report are from benchmarks performed by individuals with no potential conflict of interest in the results. The data and observations drawn from the data have been made available to each system vendor. The report is intended to be informative and instructive in state of the art benchmarking of database system architectures, and not a critical evaluation of commercial database systems.

1. INTRODUCTION

Benchmark testing is the most accurate method of evaluating the performance of a database system. Today's database systems are too complex to be satisfactorily evaluated using the techniques of analytic modeling and simulation. Previous research on the performance evaluation of database systems using modelling and benchmarking techniques have been surveyed in [BENI 84]. The survey finds that, when feasible, benchmarking a system provides the most comprehensive evaluation of system capability and performance.

1.1 Objectives

The purpose of this guide is to document the design and test of a benchmarking methodology which evaluates the performance of database management systems. Once the new benchmarking methodology was developed, it was applied to three different database systems representative of current microcomputer, minicomputer, and database machine architectures. These experiments demonstrated the viability of the methodology, and provided performance measures which characterize today's database systems under these environments. The final objective of the study was to reach conclusions, based upon the results of the benchmark experiments, which span the three architectural classes. Observations are made about the performance of each type of system architecture, rather than comparing three commercial database systems.

In an accompanying report [BENI 84], a concise survey of past research on the performance evaluation of database systems is presented, with an emphasis on previous benchmarking studies. A comprehensive benchmarking methodology for use on database systems is detailed. In this report, three benchmarks experiments on the architecturally dissimilar database systems are presented. Observations that compare performance across the system architectures are made based upon the benchmark results. In the remainder of this section the selection of the three database systems is discussed.

1.2 A Caveat

To use this document appropriately, it is necessary to understand that many aspects of the environments encompassing this guideline are changing very quickly. The microcomputer marketplace in particular, with the types of products and the pace at which they are announced and made available, is undergoing rapid transformations. In the short time since the testing phase of this study and the publication of results, the particular hardware and software configurations probably could not be duplicated due to the constant update of hardware and software. Therefore, care must be exercised in interpreting this snapshot taken of a dynamic environment, and particular results should not be heavily relied on out of this context.

1.3 Selection of Database Systems

It is highly desirable to choose systems of the same data model. Although there are several popular data models for commercial database management systems, few have been implemented on a database machine. Experimental database machines have been structured on the network data model (e.g., NEC's experiment [MAKI 82]) and the hierarchical data model (e.g., ADABAS machine). However, only the relational data model has been successfully used as a basis for a commercial database machine (e.g., Britton-Lee's IDM-500 and IDM-200). In fact, it is shown in [LUO 82] that the relational data model provides a much more efficient interface to a database machine than other data models. In this study, all the target systems will utilize the relational data model. Appendix D discusses the rationale behind the selection of the actual systems used to represent the three architectural classes for this benchmark study.

2. A BENCHMARK METHODOLOGY FOR DATABASE SYSTEMS

Managing a database requires a large, complex system made up of hardware, software, and data components. A benchmark methodology for database systems must consider a wide variety of system variables in order to fully evaluate performance. Each variable must be isolated as much as possible to allow the effects of that variable, and only that variable, to be evaluated. Because of the complex, interactive nature of database systems, it is often very difficult, if not impossible, to do this. A benchmark methodology in which a designer can identify key variables of a database system to be evaluated has been developed in [BENI 84]. In this section, a summary of this methodology is presented.

The benchmark methodology for database systems consists of three stages:

1. Benchmark Design - Establishing the system environment for the benchmark involves designing the system configuration, test data, workload, and deciding on the fixed and free variables of the benchmark studies.

2. Benchmark Execution - Performing the benchmark and collecting the performance data.

3. Benchmark Analysis - Analyzing the performance results on individual database systems and, if more than one system is benchmarked, comparing performance across several systems.

Figure 2.1 illustrates a flow chart of the methodology.

2.1 Benchmark Design

The design of a benchmark involves establishing the environment of the database system to be tested, and developing the actual tests to be performed. The four steps of the benchmark design phase are surveyed below. For a comparative benchmark over several database systems the benchmark design should be invariant over all systems. Note, however, that for the benchmark study in this report the system configurations of the three database systems are varied since the goal is to study databases in different architectures.

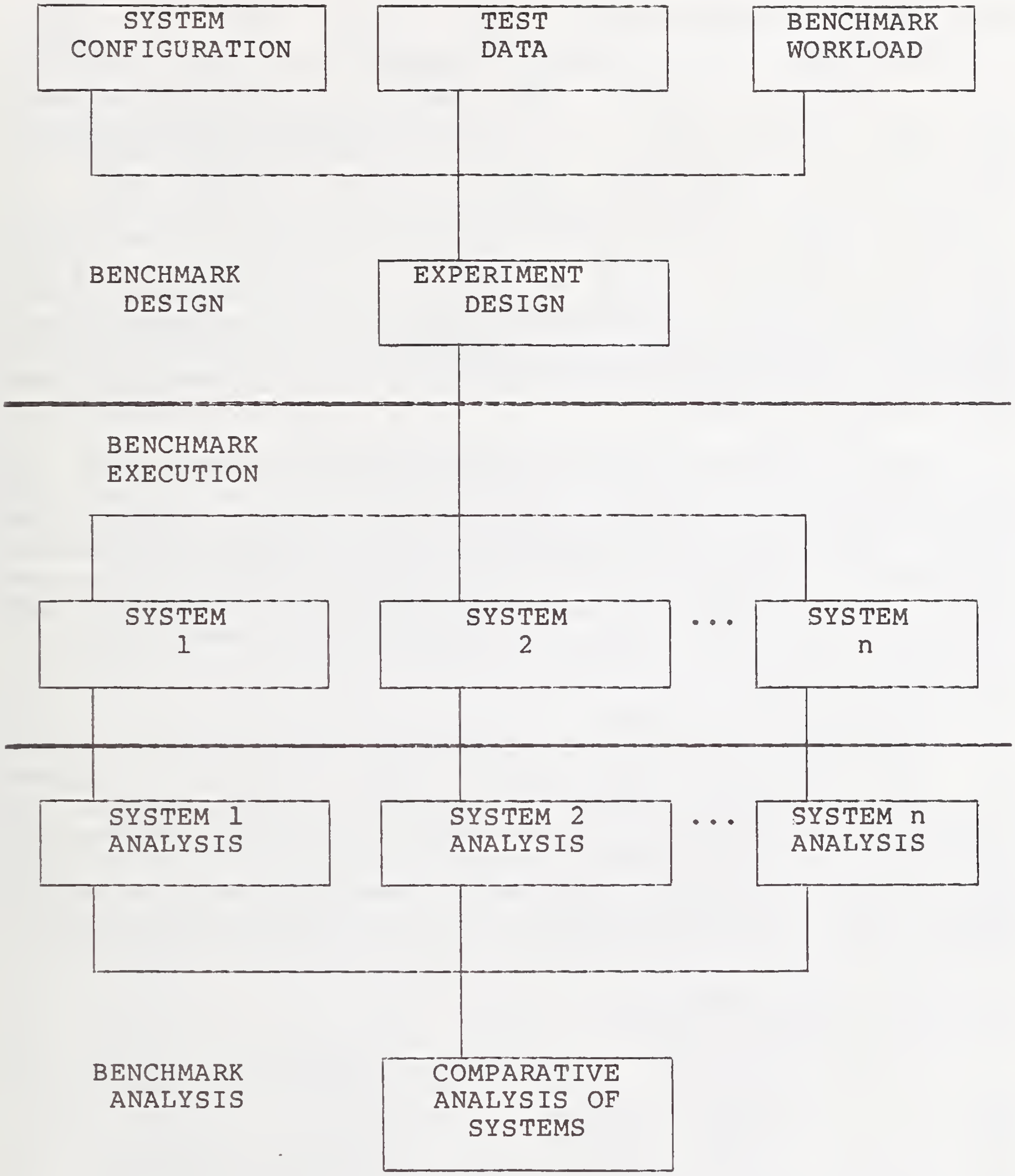


Figure 2.1: Database System Benchmark Methodology

2.1.1 System Configuration.

The hardware and software parameters, such as main memory size, the number and speed of the disk drives, operating system support for database system requirements, and load scheduling policies will be determined under this step. Often the hardware and software configuration is given. This is generally the case when the database system is to be added to an existing computing system. Also, many database systems can be installed on only one or very few types of operating systems. Cost is virtually always a factor, and for many applications it will be the primary determinant of which system configuration is actually chosen. Therefore, specifying this aspect of the environment is usually straightforward.

The parameters related to configuration that could be varied in the testing include maximum record length, the blocking factor of data in the storage system (e.g., the amount of data transferred by one disk access), the number of allowable indexes on relations, the maximum size and number of attribute values in an index, and the other types of direct access retrievals and their performance costs.

2.1.2 Test Data.

Among the parameters considered here are the test database, the background load, and the type and amount of indexing. The database on which the testing is performed can be generated using one of two methods. The traditional method has been to use an already existing database, reformating it for the benchmark needs. Recently, however, the approach of generating a synthetic database has been gaining popularity. These two methods are described and compared in [BENI 84].

2.1.3 Benchmark Workload.

A transaction load consists of several qualitative and quantitative aspects. Some of the qualitative aspects relating to transaction load are: the types of queries which occur (e.g., simple retrieval on a single relation or complex queries involving many files), the possible modes used for modification of the database (e.g., batch updates, updates in conjunction with queries), the level of user-system interaction (e.g., on-line vs. batch access), and whether or not users commonly access the same data files. Some of the quantitative aspects of the transaction load include: the percentage of time that each type of database action is performed, the average amount of data returned to a user per transaction, the average number of users, and the number of users involved in each type of transaction. Therefore, the transaction load defines not only the number of users present in the system, but also the quality and degree of activity introduced into the system by each user.

2.1.4 Experiment Design.

In this phase of the benchmark design the parameters must be considered to decide which ones will be varied in the benchmark testing. Values to be used for the parameters must also be defined. It is very important to choose values that, while within reason for the system being tested, push the system to identify it´s limitations. Among the parameters to be considered include database size, background load, number of indexes, query complexity, and number of simultaneous users.

It is also in this phase of the benchmark design that the criteria to be used for evaluation are considered. It is important to realize that the planned use of the system to be selected will have a definite relation to the main measurement criteria on which the systems are evaluated. For example, if the system is expected to be used on a heavily loaded basis and is likely to become CPU bound, system utilization or throughput should most likely be the main measurement criteria. On the other hand, if the system is more likely to be run under a light or moderate workload, response time would most likely be the most important criteria.

Benchmark experiments normally produce large amounts of output data that are too burdensome to evaluate. The final phase of a good benchmark experiment, therefore, must be a concise summary of the results obtained. This summary should point out the interesting results of the experiment and attempt to explain the reasons behind these results. A good summary will also present graphs relating testing parameters to response measures and matrices comparing results under varying variables.

2.2 Benchmark Execution

After the time consuming and complex task of designing the benchmark is completed, the next step is to execute the experiments. It would make benchmarking a much less complicated task if the benchmark could be implemented exactly as designed on each individual system to be tested. In reality, this is seldom the case. Each system has its particular design and limitations to be considered. Therefore, the benchmark has to be tailored to each specific system involved in the testing.

2.3 Benchmark Analysis

Benchmark analysis involves forming the raw performance data into graphs and tables that clearly illustrate observations and comparisons on the systems benchmarked. The analysis phase is divided into two steps.

2.3.1 Individual System Analysis.

For each system the collected data are analyzed to provide observations on the performance of the system and its environment. In this report an extensive system analysis is performed on the performance data that were gathered for the microcomputer, minicomputer, and back-end machine database systems.

2.3.2 Comparative Analysis.

If more than one system is being studied, performance data can be compared among similar systems. This analysis step should provide a basis to make validated statements as to a critical comparison between candidate database systems. Note that the goal of this report is not to make a comparative analysis among the three database systems that were benchmarked, but to analyze the characteristics of the representative database system architectures.

3. TEST DATA

For benchmark testing in this project the Department of Commerce provided a tape containing personnel data. The data on the tape came from a large application file system. All sensitive data had been eliminated from the file before it was received. The details of the file formats on the tape are contained in Appendix A.

From this data, a personnel database schema was designed. In the database design process, a core entity (PERS_DATA) of the basic personnel data attributes (e.g., SSN, NAME, SEX, BIRTH_DATE, etc.) was identified. Other entities were formed and the relationship of these entities was defined with the PERS_DATA entity. For certain attributes the data tape was incomplete because of sensitive data cleansing and empty data fields. For these attributes data values were synthesized and placed into the appropriate database relations in the database. The final schema, modeled in an Entity-Relationship Diagram, is presented in Figure 3.1.

A program was written to extract the data from the data tapes and place it into a relational representation of the database schema. The data extraction was done in two steps. First all pertinent data elements were read from the tape and written into a single first normal form relation. From the single relation the second step constructed the third normal form relations for the benchmark tests. These relations are listed in Figure 3.2 along with the record size, in bytes, of each.

Three different levels of indexing were studied in the benchmark. Level 0 contained no indexes on any of the database attributes. Level 1 provided primary indexes on the database. Unique, clustered indexes were built on all primary keys and an unclustered index was built on JOB_HISTORY.AGENCY. (The definitions for clustered and unclustered indexes can be found in the Glossary.) The database systems' ability to provide combined indexes was also tested. Level 1 included three combined indexes. Level 2 included the indexes from level 1 and added additional secondary indexes on attributes that were used for retrieval in the benchmark query set described in the next section. The specific indexes constructed on the personnel database are shown in Figure 3.3.

The different size databases were constructed by eliminating a percentage of records from the single relation form of the database. The original data from the personnel data tape formed a single relation of size approximately 56 MB (Megabytes) and containing 189,960 records. By randomly eliminating records several database sizes for the benchmark experiments were able to be formed. For the 3.5 MB database 10,500 records were retained, for the 6 MB database 20,000 records were retained, and for the 10 MB database 33,000 records were retained. Once the appropriate number of records was contained in the single relation, the the program to divide the data into multiple relations was run to build the test database.

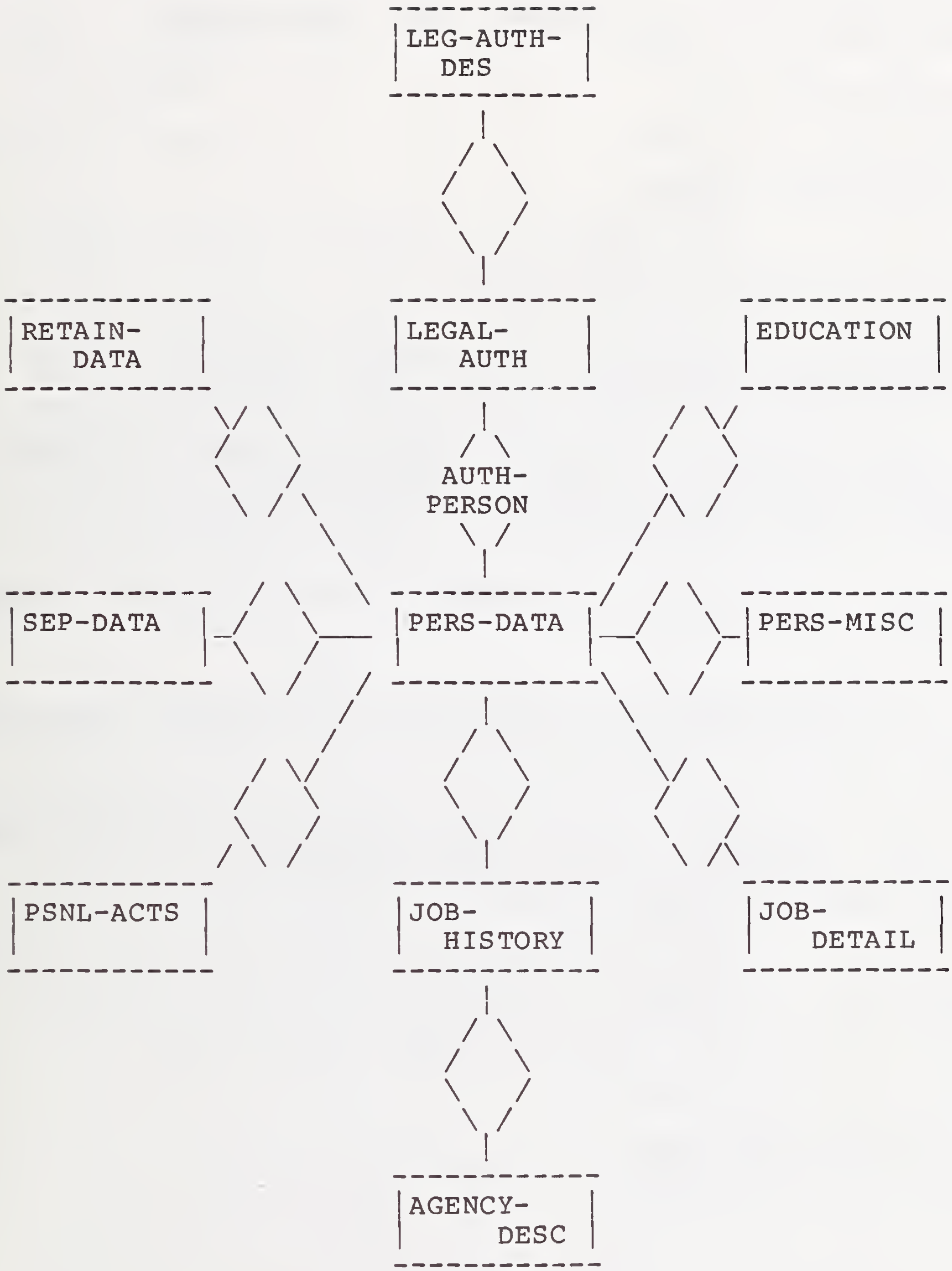


Figure 3.1: Personnel Database Schema

AGENCY_DESC(AGENCY, SUBELEMENT, NAME, DESCRIPTION)
RECORD SIZE = 33 BYTES

AUTH_PERS(SSN, CODE)
RECORD SIZE = 8 BYTES

LEGAL_AUTH(CODE, NAME)
RECORD SIZE = 22 BYTES

LEGAL_AUTH_DES(CODE, EXPLANATION)
RECORD SIZE = 45 BYTES

EDUCATION(SSN, EDUC_LEVEL, DEGREE_DATE, ACAD_DISC)
RECORD SIZE = 17 BYTES

JOB_DETAIL(SSN, PAY_DETERM, PAY_BASIS, PAY_PLAN, PAY_GRADE, CUR_GRDE_DT, CUR_OCC_DT, STEP, TENURE, PATCO, GS_EQUIV, BARG_UNIT, FLSA_EXEMPT, APPT_TYPE, UPDAT)
RECORD SIZE = 53 BYTES

JOB_HISTORY(SSN, SERV_DATE, AGENCY, SUBELEMENT, STATE, SALARY, STD_METRO, WORK_SCHED, POS_OCCUPIED)
RECORD SIZE = 25 BYTES

PERS_MISC(SSN, SUBMIT_OFF, SPEC_PRG_ID, RETIREMENT, ANNUITANT, FEGLI, PMIP, VET_PREF, VIET_VET)
RECORD SIZE = 17 BYTES

PERS_DATA(SSN, NAME, SEX, CITIZEN, BIRTH_DATE, HANDICAP, RACE, SERV_DATE, OCCUPATION, FUNCT_CLASS, LOCATION, STAT_CODE, MNGR_LEVEL, MNGRS_SSN, UPDAT)
RECORD SIZE = 85 BYTES

PSNL_ACTS(SSN, ACT_NATURE, PSNL_ACT_DT)
RECORD SIZE = 12 BYTES

RETAIN_DATA(SSN, RET_GRADE, RET_STEP, RET_PAY_PLAN)
RECORD SIZE = 11 BYTES

SEP_DATA(SSN, SEP_DATE)
RECORD SIZE = 9 BYTES

Figure 3.2: Relational Database Schema

Level 0 Indexes:

No indexes.

Level 1 Indexes:

```
Unique, clustered index on PERS_DATA(SSN)
Unique, clustered index on PERS_MISC(SSN)
Unique, clustered index on SEP_DATA(SSN)
Unique, clustered index on RETAIN_DATA(SSN)
Unique, clustered index on JOB_HISTORY(SSN, SERV_DATE)
Unique, clustered index on JOB_DETAIL(SSN)
Unique, clustered index on EDUCATION(SSN, EDUC_LEVEL)
Unique, clustered index on PSNL_ACTS(SSN)
Unique, clustered index on LEGAL_AUTH(CODE)
Unique, clustered index on LEG_AUTH_DES(CODE)
Unique, clustered index on AGENCY_DESC(AGENCY)
Unique, clustered index on AUTH_PERS(SSN, CODE)
Nonclustered index on JOB_HISTORY(AGENCY)
```

Level 2 Indexes:

```
All Level 1 indexes
Nonclustered index on RETAIN_DATA(RET_GRADE)
Nonclustered index on RETAIN_DATA(RET_PAY_PLAN)
Nonclustered index on PERS_DATA(RACE)
Nonclustered index on PERS_DATA(LOCATION)
Nonclustered index on JOB_DETAIL(PAY_GRADE)
Nonclustered index on AGENCY_DESC(SUBELEMENT)
Nonclustered index on EDUCATION(EDUC_LEVEL)
Nonclustered index on JOB_DETAIL(PATCO)
Nonclustered index on PERS_DATA(HANDICAP)
Nonclustered index on PERS_MISC(VET_PREF)
Nonclustered index on JOB_HISTORY(STATE)
Nonclustered index on EDUCATION(ACAD_DISC)
```

Figure 3.3: Three Levels of Indexes.

4. BENCHMARK WORKLOAD

4.1 Queries

A number of queries were designed to test the retrieval and update capabilities of the database systems. The queries were written in the SQL query language for the microcomputer and minicomputer database systems, and in QUEL for the database machine system. For exposition purposes the SQL formation of the queries was used in this section. The full set of QUEL queries are found in Appendix B.1. The full set of SQL queries are found in Appendix B.2.

The queries were divided into several test categories. For data retrieval ten query sets and several special purpose queries were developed to test specific database system features. Each query set contained from four to seven queries that varied in complexity based upon the number and type of conditions in the query predicate. The complexity classification developed by Cardenas [CARD 73] was used in the benchmark methodology. The range of complexities in each query set can be illustrated by examining query set 1.

Query ql-1 required retrieval of all records in the database relation.

```
ql-1: select ssn, ret_grade, ret_pay_plan
        from retain_data
```

Query ql-2 contained a single <u>atomic</u> <u>condition</u> on the relation.

```
ql-2: select ssn, ret_grade, ret_pay_plan
        from retain_data
        where ret_pay_plan = 'WG'
```

Query ql-3 contained a single <u>item</u> <u>condition</u> as a disjunction (OR) of two atomic conditions.

```
ql-3: select ssn, ret_grade, ret_pay_plan
        from retain_data
        where ret_pay_plan = 'WG'
           or ret_pay_plan = 'GM'
```

Query ql-4 contained a single <u>record</u> <u>condition</u> as a conjunction (AND) of two item conditions.

```
ql-4: select ssn, ret_grade, ret_pay_plan
```

```
        from retain_data
        where (ret_pay_plan = 'WG' or ret_pay_plan = 'GM')
          and ret_grade > '08'
```

Query q1-5 added another item condition to the record condition.

```
q1-5: select ssn, ret_grade, ret_pay_plan
        from retain_data
        where (ret_pay_plan = 'WG' or ret_pay_plan = 'GM')
          and ret_grade > '08'
          and ret_grade < '12'
```

Query q1-6 contained a single <u>query condition</u> as a disjunction (OR) of record conditions.

```
q1-6: select ssn, ret_grade, ret_pay_plan
        from retain_data
        where ((ret_pay_plan = 'WG' or ret_pay_plan = 'GM')
          and ret_grade > '08'
          and ret_grade < '12')
          or  ret_grade = '07'
```

Each query set was designed so that the complexities of the query predicates increased with increased numbers in the set. The number of records retrieved changed from one query to the next depending on whether an OR condition (increase in records) or an AND condition (decrease in records) was added.

The query sets were designed as follows. Query sets 1 through 5 tested single relation retrieval. The query sets ranged from retrieval on a small size relation (Query set 1), medium size relations (Query sets 2 and 3), and large size relations (Query sets 4 and 5). Query sets 6 through 10 tested multiple relation retrieval. Query sets 6 through 8 were two relation queries. Query set 9 was a three relation query and Query set 10 was a four relation query. The basic query sets are presented in Appendix B and are numbered for reference as qx-y, where x is the query set number and y is the number of the query in the set (e.g., q3-4). The number of bytes in the result record of each query set is given in Table 4.1.

Table 4.1: Record Size Results

QUERY SET	BYTES RETURNED
1	15
2	14
3	11
4	4
5	11
6	15
7	11
8	13
9	18
10	37

To test the sorting facility, Query sets 1 through 4 were modified by adding an 'ORDER BY' clause on an appropriate attribute. These queries are designated with an 'o' in this report (e.g., qo2-3). Appendix B contains the details of these queries.

To test aggregates query sets 4 and 5 were modified by adding the 'COUNT' aggregate function in the output list. These queries are designated with an 'x' in this report (e.g., qx5-2) and are presented in Appendix B.

Each database system was tested for three special cases of data retrieval. The tests on the microcomputer system were slightly modified because of system limitations.

Special Case 1: Arrangement of Conditions

The performance of the following two queries was compared. The only difference between the queries was the arrangement of the query conditions.

```
scl-1:  select pers_data.ssn, educ_level, birth_date, vet_pref
                 from pers_data, education, pers_misc
                 where pers_data.ssn = education.ssn
                 and education.ssn = pers_misc.ssn
                 and pers_misc.ssn = 300378541

scl-2:  select pers_data.ssn, educ_level, birth_date, vet_pref
                 from pers_data, education, pers_misc
                 where pers_misc.ssn = 300378541
                 and pers_data.ssn = education.ssn
                 and education.ssn = pers_misc.ssn
```

Special Case 2: Implicit vs. Explicit Conditions

With the same query the performance was tested when the join conditions were made explicit.

```
sc2-1:  select pers_data.ssn, educ_level, birth_date, vet_pref
               from pers_data, education, pers_misc
               where pers_data.ssn = education.ssn
               and education.ssn = pers_misc.ssn
               and pers_misc.ssn = 300378541

sc2-2:  select pers_data.ssn, educ_level, birth_date, vet_pref
               from pers_data, education, pers_misc
               where pers_data.ssn = 300378541
               and education.ssn = 300378541
               and pers_misc.ssn = 300378541
```

Special Case 3: Join Optimization

A test was made to estimate the performance of a two relation join with a sort/merge optimization technique. This performance was compared with the query performance as found with the standard join optimization method found in the database system.

```
sc3-1:  select retain_data.ssn, ret_grade, barg_unit
               from retain_data, job_detail
               where retain_data.ssn = job_detail.ssn
               and (patco = 'T' or patco = 'O')
```

A sort/merge optimization of this query was approximated by running the following three separate queries. Note that query sc3-3 uses a temporary relation 'templ' that was previously created.

```
sc3-2:  select ssn, ret_grade
               from retain_data
               order by ssn

sc3-3:  insert into templ
               select ssn, barg_unit
               from job_detail
               where (patco = 'T' or patco = 'O')

sc3-4:  select ssn, barg_unit
               from templ
```

In addition to the retrieval queries, the performance of several update commands was tested. Representative insertions (e.g., qI-1), deletions (e.g., qD-1), and modifications (e.g., qM-2) were designed on the personnel database. These updates are found in Appendix B.

4.2 Job Scripts

Benchmark workloads were generated by defining job scripts for each benchmark run. The job script is a file of query numbers. For example, a job script for a benchmark run could be {q1-1, qx5-3, q9-4, qI-3, q4-5}.

The job script file is read into a program on the front-end or host computer called 'runner'. Runner executes the queries in job script order on the database system. Runner records statistics on the query's performance in the system. Times were recorded to the sixtieth of a second to three decimal places for the minicomputer and database machine benchmarks. The times recorded for the microcomputer tests were to the second. The statistics gathered were:

```
parse   - Query parse time
execute - Query execution time
first   - Time until first record is retrieved
last    - Time until last record is retrieved
records - Number of records returned
```

The runner algorithm can be outlined as follows:

```
Algorithm runner:

Begin

  Read Job-script-file into query-array until EOF;

  Open database system;

  While (query-array not empty) do

    Read next query from query-array;
    Send query to the database system;
    Parse query;
    Execute query;
    Record time statistics on:
        time-to-parse
        time-to-execute
        time-to-first
        time-to-last
        record-count ;
    Print gathered statistics;

  End while;

  Close database system;

End of Algorithm.
```

For each benchmark test a job script was defined and the ´runner´ was executed on the different database systems. Statistics for each query in the script were printed in a convenient format. For multi-user tests and background load tests on the databases multiple copies of runner were run simultaneously on separate job scripts and the statistics gathered on each.

5. EXPERIMENT DESIGN

5.1 Performance Measures

For this project response time was selected as the primary performance measure. The ´runner´ program provided easily accessible timings from which several types of query response times could be calculated. Other potential performance measures, such as throughput and utilization, while useful, were not considered as important. Throughput in a single user environment could be obtained from response time statistics since ´runner´ initiates a query in the job script as soon as its predecessor completes. Throughput for the multi-user portion of the study could be estimated by timing the completion of all job scripts and dividing by the number of queries completed. No software or hardware monitors were available for measuring utilizations in this study.

The ´runner´ algorithm provided several timing statistics for each query as soon as the query completed. The statistics that were used primarily in the analysis are the time-to-first-record (TF) value and the time-to-last-record (TL) value. The time-to-first statistic measured the time from when ´runner´ called the database system with the query until the first result record was received at the host system. Time-to-first-record is independent of the number of records retrieved in the query and is free of potential input/output delays. The time-to-last statistic measured the total response time of the query from initiation until the last record was retrieved.

As a general policy for all benchmark studies, if an individual query ran for more than 30 minutes, the run was halted and no data were gathered for that query. (Some exceptions to this policy can be observed in the result data.) In the data tables in Appendix C dashes (---) denote a timed-out query. This policy was necessary since a number of queries required considerable time to complete under some of the test conditions.

5.2 Experiment Variables

A number of important benchmark tests could be performed on the personnel database by selecting and varying one or more dependent database variables. The following analysis variables were selected for the benchmark tests.

1. Database Size - The following database sizes were studied on the database systems.

Table 5.1: Database Sizes

SIZE	NUMBER OF RECORDS IN SINGLE RELATION
56 MB	189,960
10 MB	33,000
6 MB	20,000
3.5 MB	10,500

   Most tests were performed on the 3.5 MB, 6 MB, and 10 MB databases. Several additional benchmark tests were on an 8 MB database.

2. Query Complexity - Two factors were considered in determining query complexity. Greater complexity of the query predicate lead to increased parsing time and increased the potential for query optimization. Within each query set, the predicate complexity was increased by adding additional conditions (i.e., higher numbered queries). Second, the number of relations involved in a query indicated query complexity. More complex join operations were required between relations.

3. Records Retrieved - The time-to-last-record (TL) depended greatly upon the number of records in the query result.

4. Order of Query Execution - The different database systems used internal memory as buffers for the storage of needed indexes and intermediate results during query execution. The effect of the buffer memory on the order of query execution was tested. Job scripts formed of similar queries were executed sequentially (e.g., q2-1, q2-2, q2-3, q2-4, q2-5) to investigate whether any efficiencies occur because of the buffer. For example, a needed index may already be in the buffer. The statistics for the queries run

in order were compared with the queries run in a random order.

5. Indexing - As described in section 3, Test Data, three levels of indexing were defined on the database: 0) no indexes, 1) primary key indexes, and 2) primary key indexes plus several secondary key indexes. Tests were run varying the different index levels.

6. Sorting - Sorting performance was tested by adding ´order by´ clauses to certain query sets. By comparing sorted and unsorted queries, the sorting performance in the different database systems could be determined.

7. Aggregation Functions - Two query sets were modified by adding an aggregation function, ´count´, to the output list.

8. Number of Users - Multiple users contend for database system resources. This tends to increase the response time and increase the throughput. On the minicomputer and database machine systems, one, two, and three user environments were tested. Each user ran a separate job script. To study contention, tests were run in which each user ran an identical job script. Other tests included different job script mixes.

9. Background Load - Tests were run varying the non-database jobs in the host computer system. The number of jobs could be increased and the type of jobs in the background could be varied. Background jobs could be designed as CPU or I/O intensive jobs. Tests could determine the effect of these jobs on the performance of the database queries. By measuring the performance of the background jobs under different query loads, the effect of database jobs on the background jobs could also be studied. This was termed a reverse benchmark.

The benchmark analyses in the next three sections present performance data for the benchmark tests that resulted from varying combinations of the above experimental variables. Additional tests in the benchmarks include the special cases that were described in the workload section. These studies included: 1) different orders of query predicates (i.e., different arrangement of conditions), 2) implicit join conditions vs. explicit join conditions, and 3) sort/merge join technique in join optimization.

6. MICROCOMPUTER BENCHMARK

6.1 System Configuration and Benchmark Execution

Details of the representative microcomputer system configuration, implementation, and benchmark execution are discussed in Appendix D.

6.2 Benchmark Analysis

There are four parts in this section: single relation queries, multiple relation queries, updates, and special case queries. A complete set of performance data for the microcomputer benchmark is found in Appendix C.1.

6.2.1 Single Relation Queries.

Table MICRO.1 lists the result sizes, in number of records retrieved, for all query sets. Table MICRO.2 contains the time-to-first and time-to-last response times, respectively, for different index levels in single relation queries. (Time-to-first was not measured for all of the queries.) Note that for the microcomputer benchmark an index level 3 was added. This level added secondary indexes on every attribute in the database.

"Response Time vs. Query Complexity"

Figures 6.1, 6.2, and 6.3 show the dependency of the response times on query complexity and the amount of data retrieved. Figure 6.1 shows that the time-to-first generally decreased as the complexity of the query decreased for single relation queries with and without indexes. An intuitive explanation is that when the query complexity decreases, more records are retrieved. If records are ´uniformly´ distributed in the relation, the time for reaching the first record should be shorter. This result was valid for all single relation query sets. Note that the gaps after query q2-3 show where ´OR´ clauses were added to the predicates. This increased the number of records retrieved by the query and affected both time-to-first and time-to-last performance. To clearly show the effect of adding ´AND´ clauses to the query predicates, the line when the ´OR´ operations were added has not been connected.

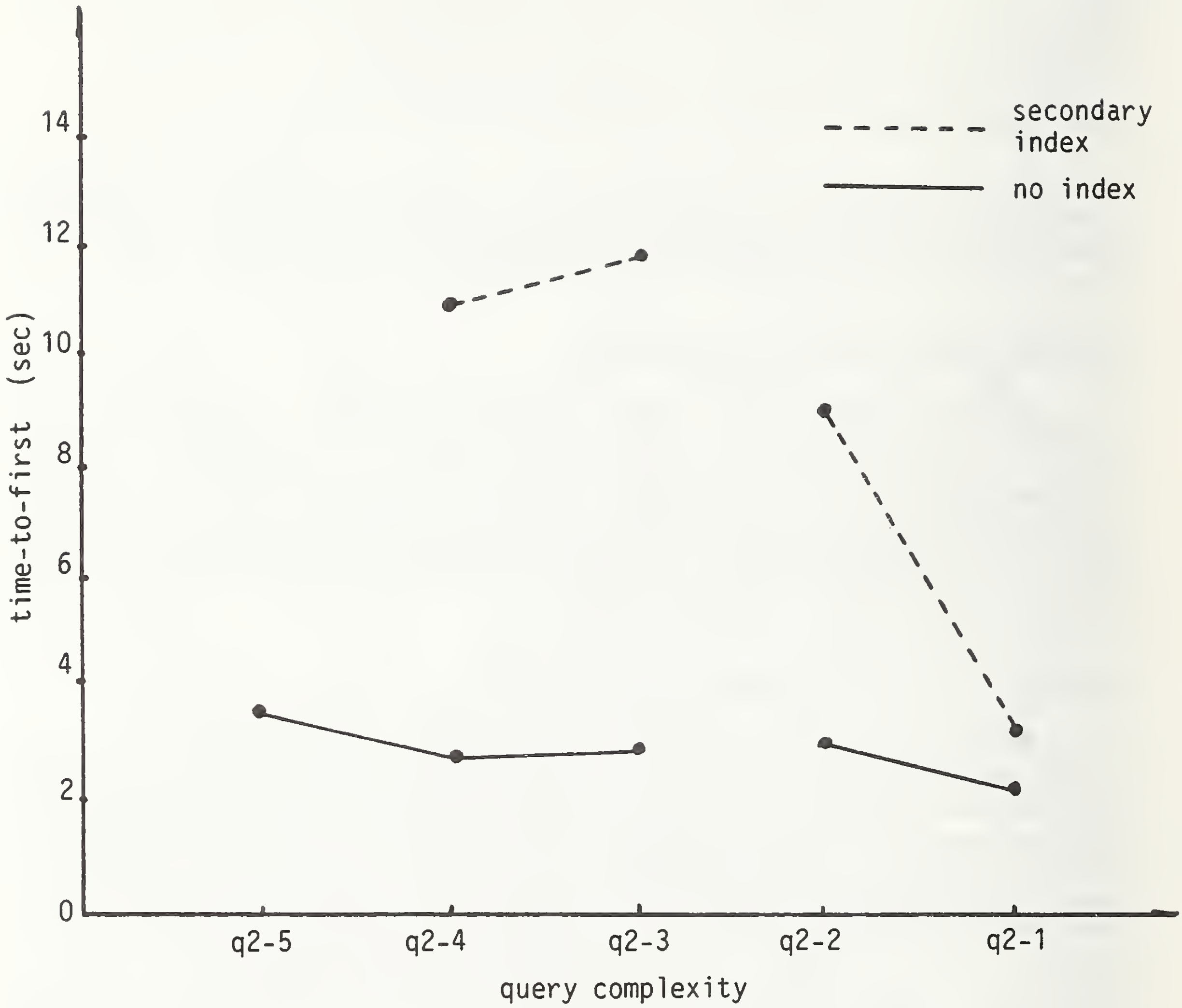


Figure 6.1: Query Complexity vs. Time to First Record.

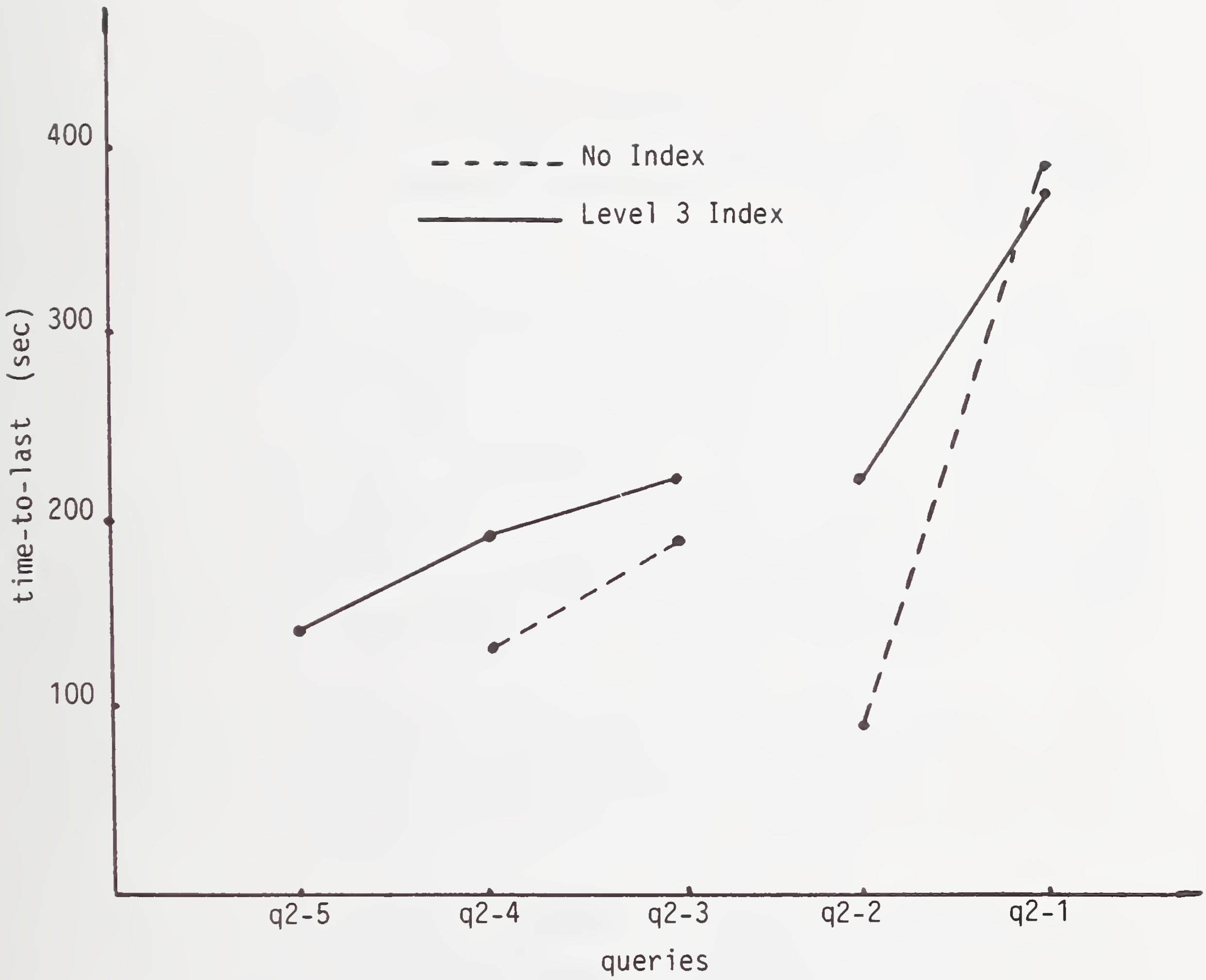


Figure 6.2: Query Complexity vs. Time to Last Record.

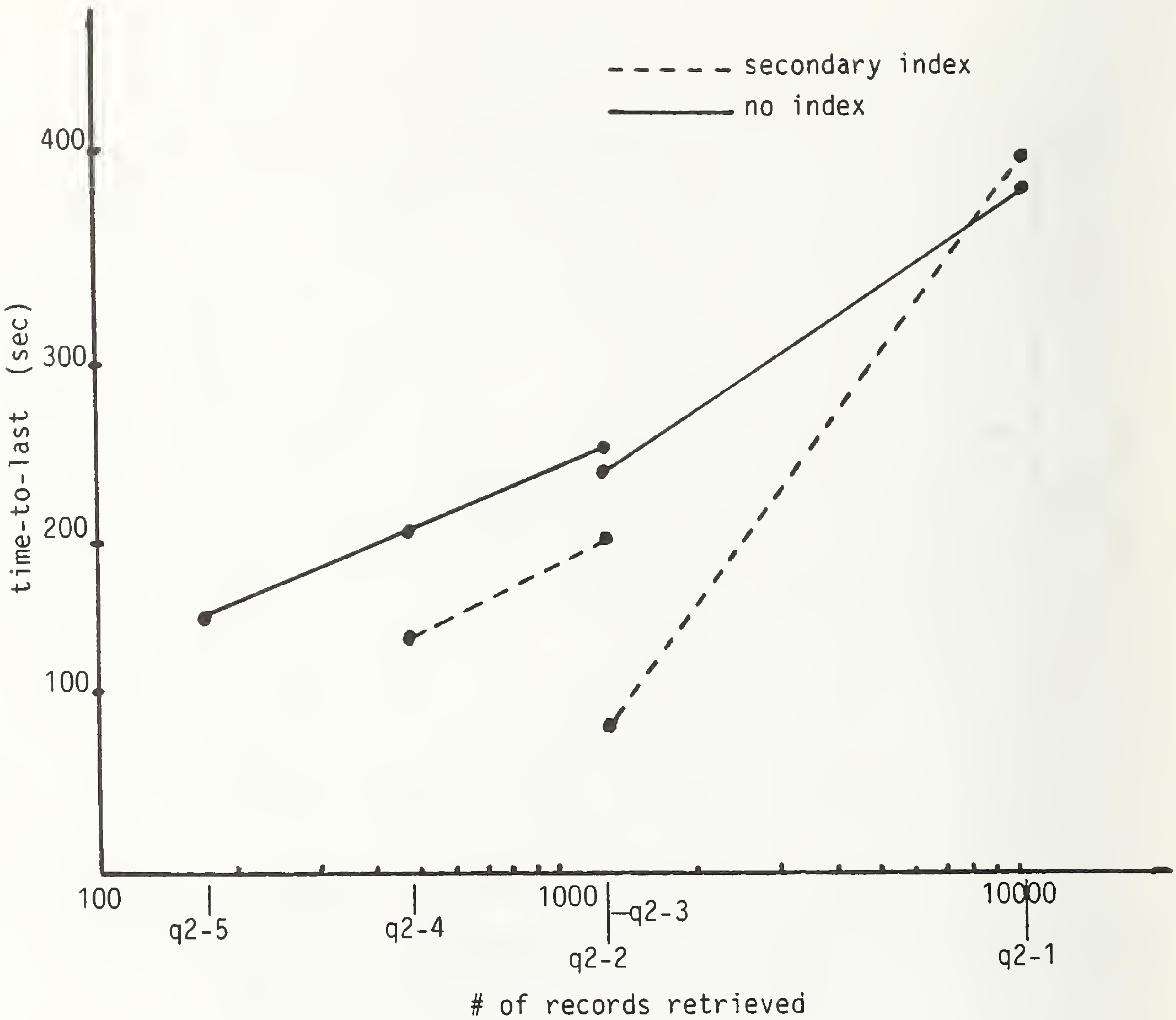


Figure 6.3: Records Retrieved vs. Time to Last Record.

On the other hand, Figure 6.2 shows that time-to-last increased as the complexity of the queries decreased for all database sizes. Here the factor which influenced response time to the last record was the total number of records retrieved, which is roughly inversely proportional to the query complexity. Adding the 'OR' clause to query q2-2 increased the number of records retrieved and accordingly increased the time-to-last for those queries over the previous query in the set. In Figure 6.3, the time-to-last increased as the number of records retrieved increased. From q2-2 to q2-1 and from q2-5 to q2-3, as can be seen in Figure 6.3, I/O delay was the major factor which caused the increase of time-to-last. Query q2-5 did not complete with secondary indexes because of index pointers overflowing the workspace.

"Effect of Indexing"

With respect to the comparison between queries with and without indexes, the performance difference can be attributed to the fact that the queries with indexes included additional time for index access and optimization before the first record could be returned. The benefit, however, was that the system could retrieve records directly without performing a sequential search through the database files. As can also be seen in Figure 6.4, query set 2 with indexes had a better performance than without indexes for time-to-last. Over all query sets, it was found that when indexes could be used effectively, the time-to-last was reduced for single relation queries.

For most queries, the time-to-first was, however, not improved by indexing. Indexing did not even benefit the time-to-last for some queries. This was due to a number of factors. For the microcomputer data the following observations could be made:

1. **Hit Ratio.** The hit ratio refers to the percentage of records retrieved from a relation. Indexes are useful for retrieving a relatively low percentage of the file because they can avoid the searching of the entire relation sequentially. If the time-to-last performance in Table MICRO.2 is examined, it can be seen that the high hit ratio penalized the performance of queries 1-2, 1-3, 1-4, and 1-5. The hit ratios of these queries were determined from Table MICRO.1 as 55.6%, 56.5%, 36.7%, and 13.9% respectively.

2. **Disjunctive Index.** The use of a disjunctive condition (terms connected by 'OR') in a query can sometimes make index processing more difficult. It is well known that if one of the terms in the disjunction is not indexed, then any other index in the condition is not useful. However, if all the terms are indexed, or if two terms in a disjunction use the same index, then the performance depends on the particular implementation. In the microcomputer benchmarks, a performance penality for disjunctive indexes was observed on queries 2-3, 2-4, 2-5, and 2-7, for example.

3. **Range Index.** The use of an index to solve a range query may have a similar effect as the disjunctive index. This effect was observed on queries 3-4, 3-5, and 5-4. A combination of the disjunctive clauses and range clauses and their effects were found in queries 3-6 and 5-6.

"Effect of Aggregate Functions"

The 'count' was used to show how aggregate functions work under different system variables. The result is shown in Figures 6.4 and 6.5. The response times are in Table MICRO.3. Similar to Figure 6.3, the time-to-last increased with the number of records retrieved. The queries with the 'count' function actually had better time-to-last performance. This was probably because only the count result, not the result records retrieved, had to be produced. The conclusion above could be found in both queries using an index (Figure 6.4) and queries not using an index (Figure 6.5).

6.2.2 Multiple Relation Queries.

Time-to-last for join queries is plotted in Figure 6.6. Table MICRO.4 contains the data for multiple relation queries. As expected, there was a large difference between the queries using indexes and the ones using no index. The larger difference in multiple relation joins, compared with single relation queries, seemed to result from the 'nested loop' strategy used in the microcomputer database system that has an exponential response time.

Because of the large number of records retrieved in query sets 7 and 6, time-to-last data for these queries were unable to be found without using indexes. In fact, it was found that the time-to-last for some queries was more than 15 hours, by letting several run without aborting them. After that point, the UNIX system would terminate the

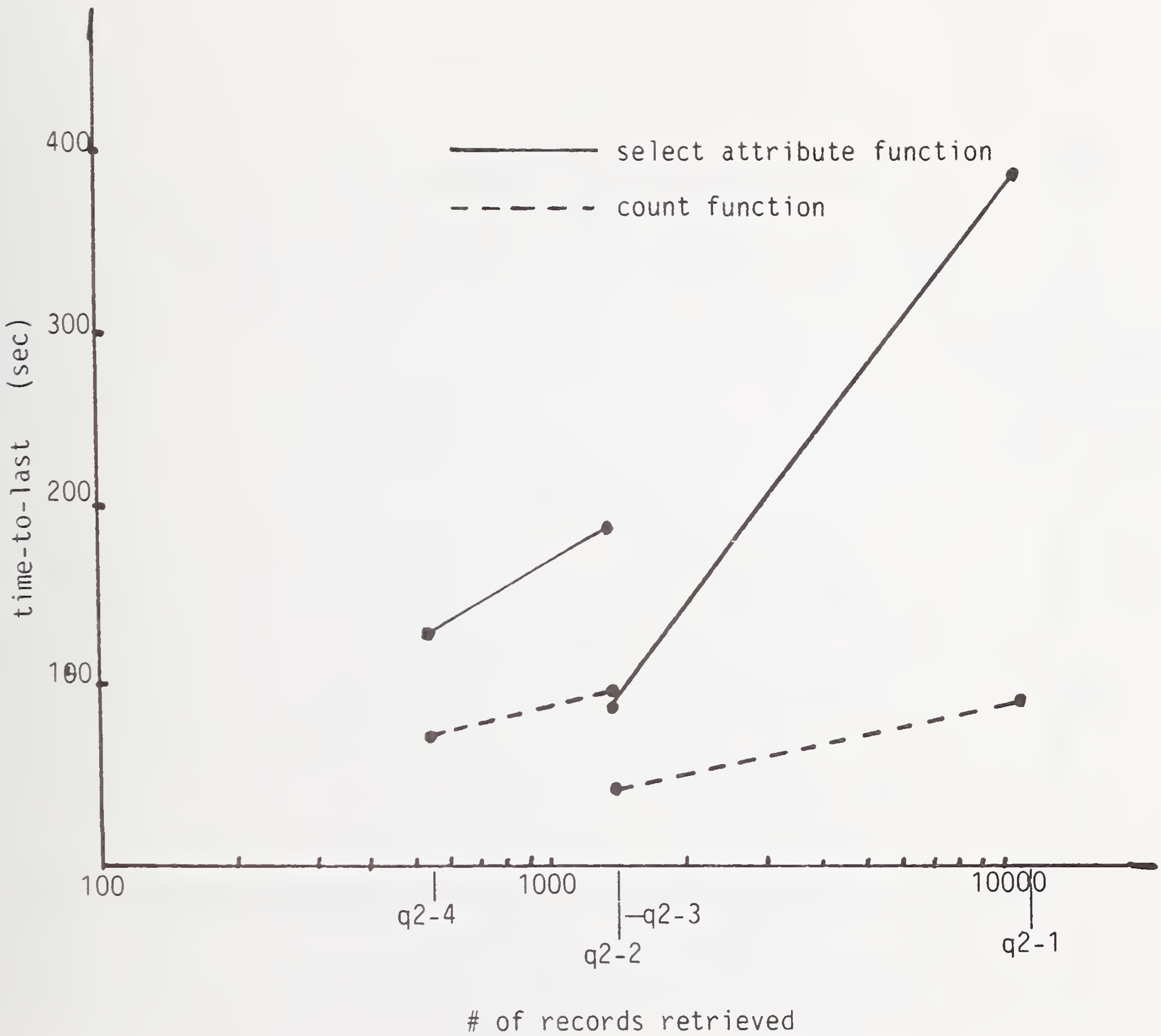


Figure 6.4: Single Relation Index Tests (Level 3 Indexes).

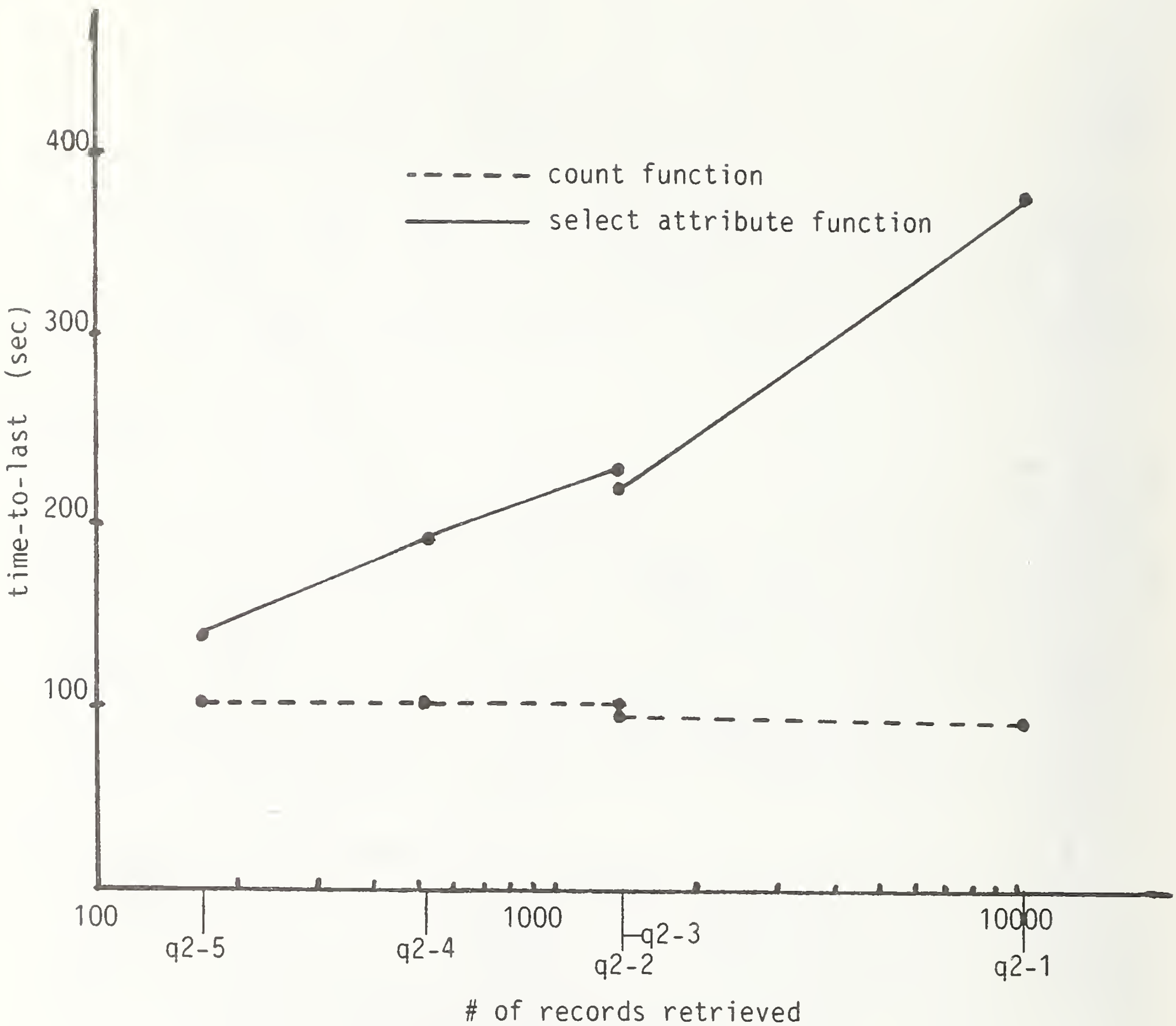


Figure 6.5: Aggregate Function Test (No Indexes).

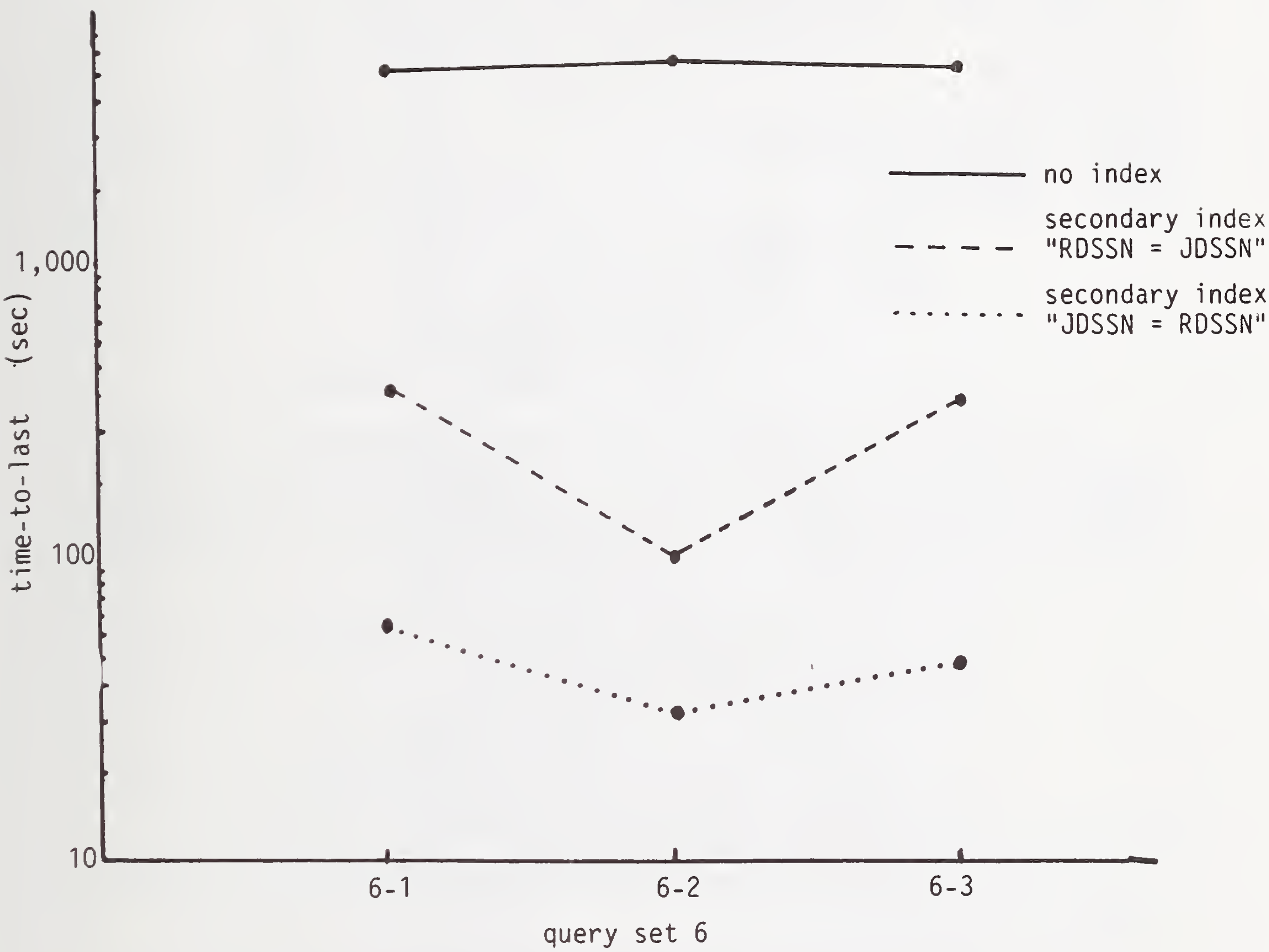


Figure 6.6: Multiple Relation Index Tests.

process and ´autologout´ the query.

In Figure 6.6, the difference between queries using and not using indexes shows the effectiveness of index access for two-relation joins. The effect of changing the order of join clauses is discussed in the section on special case queries.

The microcomputer database system did not support joins of more than two relations. For query set 9, the queries were changed to nested queries as shown below;

query q9-1:

```
select  PDSSN, Birth_Date
from    Pers_Data
where   PDSSN in
            select  ESSN
            from    Education
            where   ESSN in
                        select  PMSSN
                        from    Pers_Misc
                        where   PMSSN < ´202000000´;
```

To prevent the workspace from overflowing in a nested query, it was necessary to restrict the size of records retrieved from the ´inner´ query. In Table 6.1, the response time for queries using secondary indexes shows improvement in the nested queries.

Table 6.1 Index Effect in Nested Queries.

Response Time in Sec.		
Query	q9-2	q9-3
Level 1 Index	577	597
Level 3 Index	46.	497

"Special Studies"

The special cases used here are not the same as discussed in Section 4. Queries scl-1, scl-2, sc2-1, sc2-2, sc3-1, and sc3-2 are shown below. The benchmark for each case is discussed in turn.

```
scl-1:  select  ESSN, Educ_Level
        from    Education
        where   ESSN in
                select  PMSSN
```

```
                from    Pers_Misc
                where PMSSN < '202000000';

scl-2:  select  ESSN, Educ_Level
        from    Education, Pers_Misc
        where   PMSSN = ESSN
          and   PMSSN < '202000000';

sc2-1   select  RDSSN, Ret_Grade, Barg_Unit
        from    Retain_Dat, Job_Detail
        where   RDSSN = JDSSN;

sc2-2   select  RDSSN, Ret_Grade, Barg_Unit
        from    Retain_Dat, Job_Detail
        where   JDSSN = RDSSN;

sc3-1   select  (all)
        from    Pers_Data
        where   PDSSN in
                select  ESSN
                from    Education
                where   ESSN in
                        select  PMSSN
                        from    Pers_Misc
                        where   PMSSN < '202000000';

sc3-2   select  (all)
        from    Pers_Misc
        where   PMSSN in
                select  PDSSN
                from    Pers_Data
                where   PDSSN in
                        select  ESSN
                        from    Education
                        where   ESSN < '202000000';
```

"Join vs. Nested Join"

In SQL it is possible to write a join query as a nested query. The result of the benchmark of the two equivalent queries scl-1 and scl-2 is shown in Table 6.2 below.

Table 6.2: Nested Join Tests

Response Time in Sec.			
No Index		Level 3 Index	
scl-1	scl-2	scl-1	scl-2
636	15,167	56	91

The result showed the nested query had faster response time than the flat join. But this did not mean the nested query was better than the select-join query. It should be noted that there were two basic problems for nested queries. First, there was not enough workspace to hold the temporary data generated after processing the inner query. Second, the nested query could not select the attributes from the relation accessed in the inner query.

"Effect of the Join Sequence"

The second special test examined the effect of joining two relations in different orders. As shown in queries sc2-1 and sc2-2, the join order of query q6-1 was reversed for the tests. The result is summarized in Table 6.3 below.

Table 6.3: Join Sequence Test

	sc2-1	sc2-2
6-1	407	51
6-2	106	21
6-3	390	44

The above table shows the effect of changing the order of the comparison "RDSSN = JDSSN". There were 74 records in Retain_Data and 10,500 in Job_Detail. In sc2-2, the order was changed to "JDSSN = RDSSN" and it was found that the time-to-last dropped sharply from 407" to 51" and 390" to 44". As shown in Figure 6.6, the dashed line stands for the order "RDSSN = JDSSN" and the dotted line stands for "JDSSN = RDSSN". The size of the smaller relation, Retain_Data, was the major factor that helped to reduce the access time for matching records in the two-relation join.

"Order of Query Nesting"

The last special test examined the effect of the different order of query nesting. The result is shown in Table 6.4 below.

Table 6.4: Order of Query Blocks

No Index		Level 3 Index	
sc3-1	sc3-2	sc3-1	sc3-2
133	150	66.161	

In the nested query, the change of the order of nested blocks affected the performance. This indicated that the nested blocks should be arranged so that the smaller relations are handled first in the outer blocks and the larger relations in the inner blocks for best performance.

6.2.3 Updates.

As the update data in Table MICRO.5 was examined, it was observed that the index helped the modification and delete queries. The index on the primary key improved searching of the target records and reduced the response time. Inserting new records into an indexed relation required the indexes to be updated. The index update caused the insertion to be slower in this case.

6.3 Summary

The following points summarize the major results of the benchmark study of the microcomputer database system architecture. These points can be used as guidelines for evaluating performance of microcomputer database systems. A comparison of performance over the three tested architectures is presented in Section 9.

1. In general, query complexity was inversely proportional to the number of records retrieved. Without using indexes, the time-to-first-record was greater when fewer records were retrieved, since the system took longer to find the first record. The time-to-last-record was greater when a larger number of records were retrieved, due to the time required to access and transmit additional records.

2. Indexing reduced response time if the index could be used effectively in the query access strategy. Several types of queries were identified that did not allow the index to be used effectively. These cases included queries with high hit ratios, queries with disjunctive conditions, and queries with range conditions.

3. Sorting and aggregation functions added significant overhead to query response time.

4. Multiple relation queries (join queries) required considerably more time to complete than single relation queries. The use of primary key indexes and secondary key indexes were found to be essential for the execution of multiple relation queries. Work space size proved to be the major constraint on the ability to run multiple relation queries on the microcomputer database system. The storage of intermediate results and indexes quickly overflowed the available work space and caused a system abort of the query.

5. Each SQL query block in the microcomputer database system could only handle two relations. This restriction forced the use of nested block queries with intermediate results. Special case testing showed that the nested blocks should be arranged so that the smaller relations should be handled first in the outer blocks and the larger relations should be placed in the inner blocks for best performance.

6. The writing of the join conditions had a significant effect on performance. In a join condition such as (JDSSN = RDSSN), the attribute of the larger relation should be listed first for improved performance.

7. Update performance was significantly improved by having indexes in a relation.

7. MINICOMPUTER BENCHMARK

7.1 System Configuration and Benchmark Execution

Details of the representative minicomputer DBMS system configuration, implementation, and benchmark execution are discussed in Appendix D.

7.2 Benchmark Analysis

In this section a summary of the results obtained from executing a benchmark on the minicomputer system is presented. The discussion of results is divided into five sections: single relation queries, multiple relation queries, update queries, multiple user results, and background load results. Observations were made on the performance of the minicomputer database system based upon the analysis of the result data. The measures of time-to-first-record and time-to-last-record were used as the primary performance statistics. All minicomputer data tables and graphs are contained in Appendix C.2.

There is no claim to present here a critical analysis of the performance of any commercial minicomputer database system. Such an objective could only be achieved by running an extensive mix of benchmarks under a wide range of environments, which is beyond the scope of this report.

7.2.1 Single Relation Queries.

"Response Time vs. Query Complexity"

The dependency of the response times on query complexity and the amount of data retrieved is shown first. Table MINI.1 lists the result sizes, in number of records retrieved, for each of the single relation queries. This table matches the result size tables for the microcomputer and database machine tables for all database sizes. Table MINI.2 contains the time-to-first and time-to-last response times for the queries with level 1 indexing. Figure 7.1 shows that the time-to-first generally decreased as the complexity of the query decreased. This observation does not differ significantly for different database sizes of 3.5 MB, 6 MB and 10 MB. An intuitive explanation is that when the query complexity decreases, more records are retrieved. If records are ´uniformly´ distributed in the relation, the time for reaching the first record should be shorter. The identical result was found for the microcomputer and database machine environments. Obviously, this result is not valid if indexes are used to process the query. Note that the gaps after queries q1.6 and q1.3 show where 'OR' clauses were added to the predicates. This increased the number of records retrieved by the query and affects both time-to-first and time-to-last performance. To clearly show the effect of adding 'AND' clauses to the query predicates, the line when the 'OR' operations were added has not been connected.

On the other hand, Figure 7.2 shows that time-to-last increased as the complexity of the queries decreased for all database sizes. Here the factor which influenced response time to the last record was the total number of records retrieved, which is inversely proportional to the query complexity. Adding the 'OR' clause to queries q1.6 and q1.3 increased the number of records retrieved and accordingly increased the time-to-last for those queries over the previous queries in the set.

Figure 7.3 further reinforces this observation. When the time-to-last was plotted against the number of records retrieved, the curve showed an almost linear behavior until the records retrieved approached 10,000.

"Response Time vs. Indexing"

The benchmark tests on single relation indexing lead to the observation that the selection of secondary key indexes provided significant performance advantages for queries that contained selections on indexed attributes. Figures 7.4 and 7.5 illustrate this point using query set 5. The result data for the index tests are in Tables MINI.3 and MINI.4. In Figure 7.4 the performance of query set 5, under no indexes and level 2 indexes (secondary key indexes added), was studied for the 6 MB database. The EDUCATION relation had secondary key indexes on attributes ACAD_DISC and EDUC_LEVEL. In queries q5.1, q5.2, and q5.3 the indexed attributes were not used in the query conditions. Thus, there was no significant performance difference between the levels of indexing. In queries q5.4 and q5.5, however, predicates were added that utilized the two indexed attributes. The improved performance of the queries with index level 2 was apparent.

The advantage of indexing was reinforced by Figure 7.5 where the performance improvement of queries q5.4 and q5.5 is shown under all three index levels. This improvement was lost, however, when the query was made more complex by

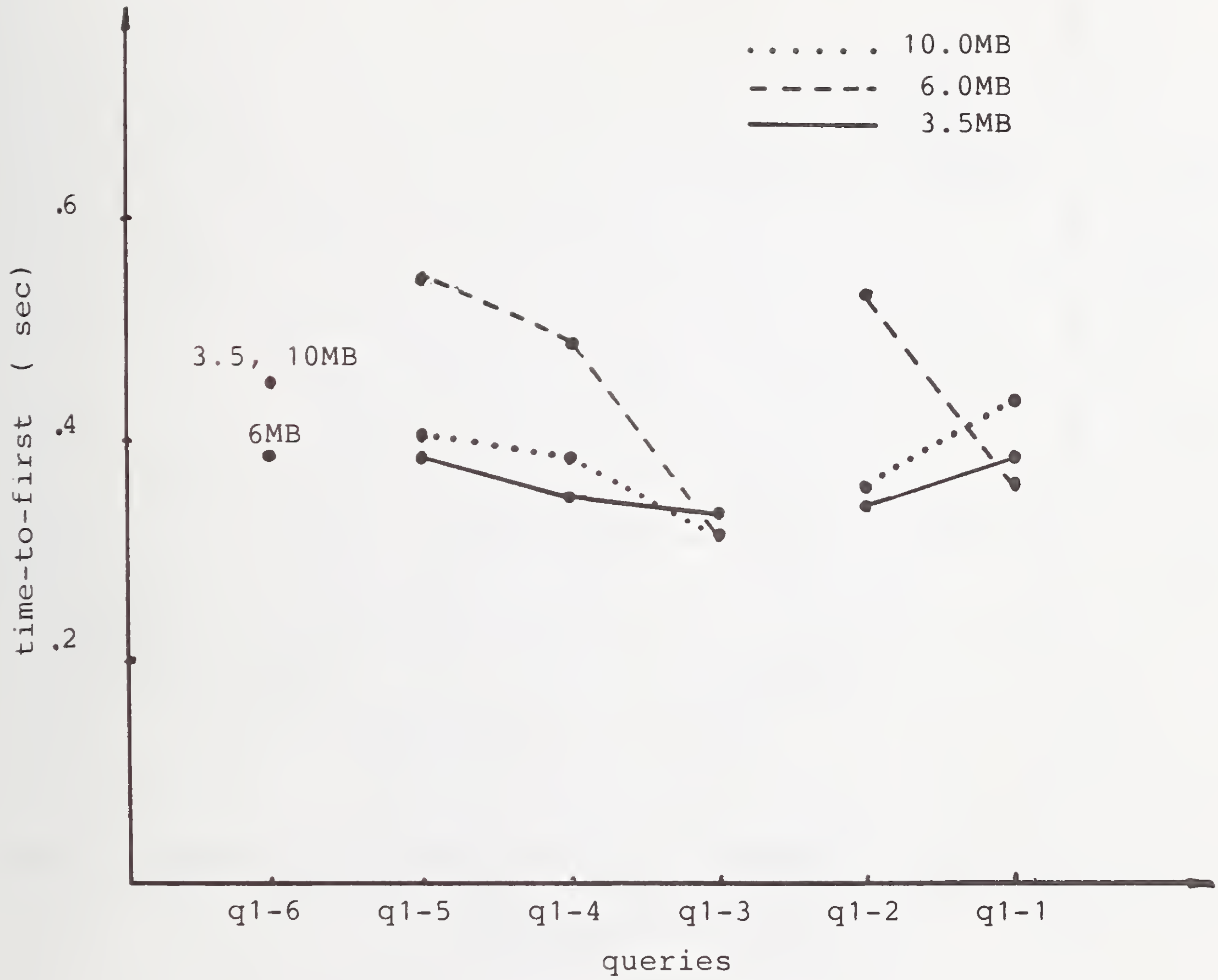


Figure 7.1: Query Complexity vs. Time to First Record.

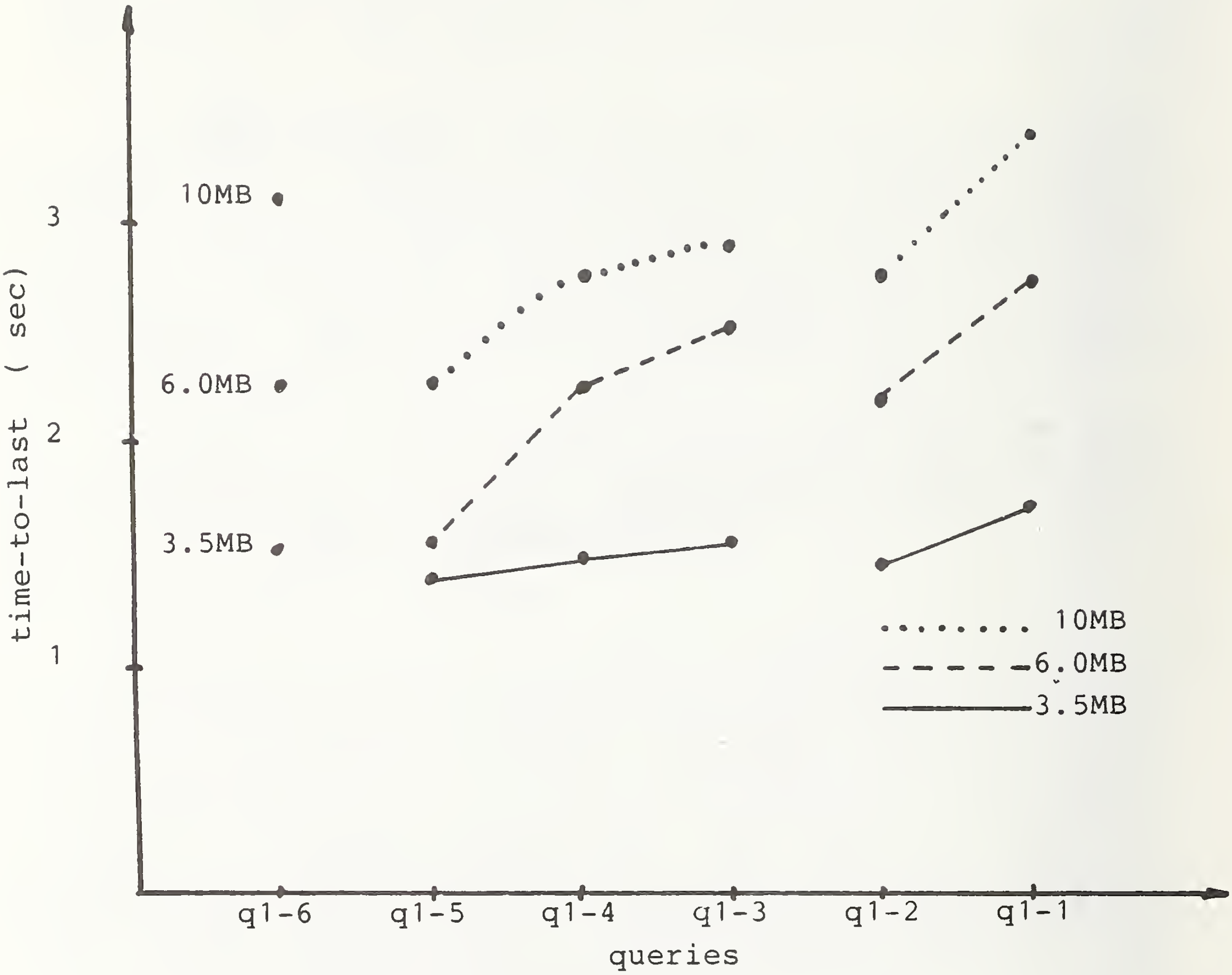


Figure 7.2: Query Complexity vs. Time to Last Record.

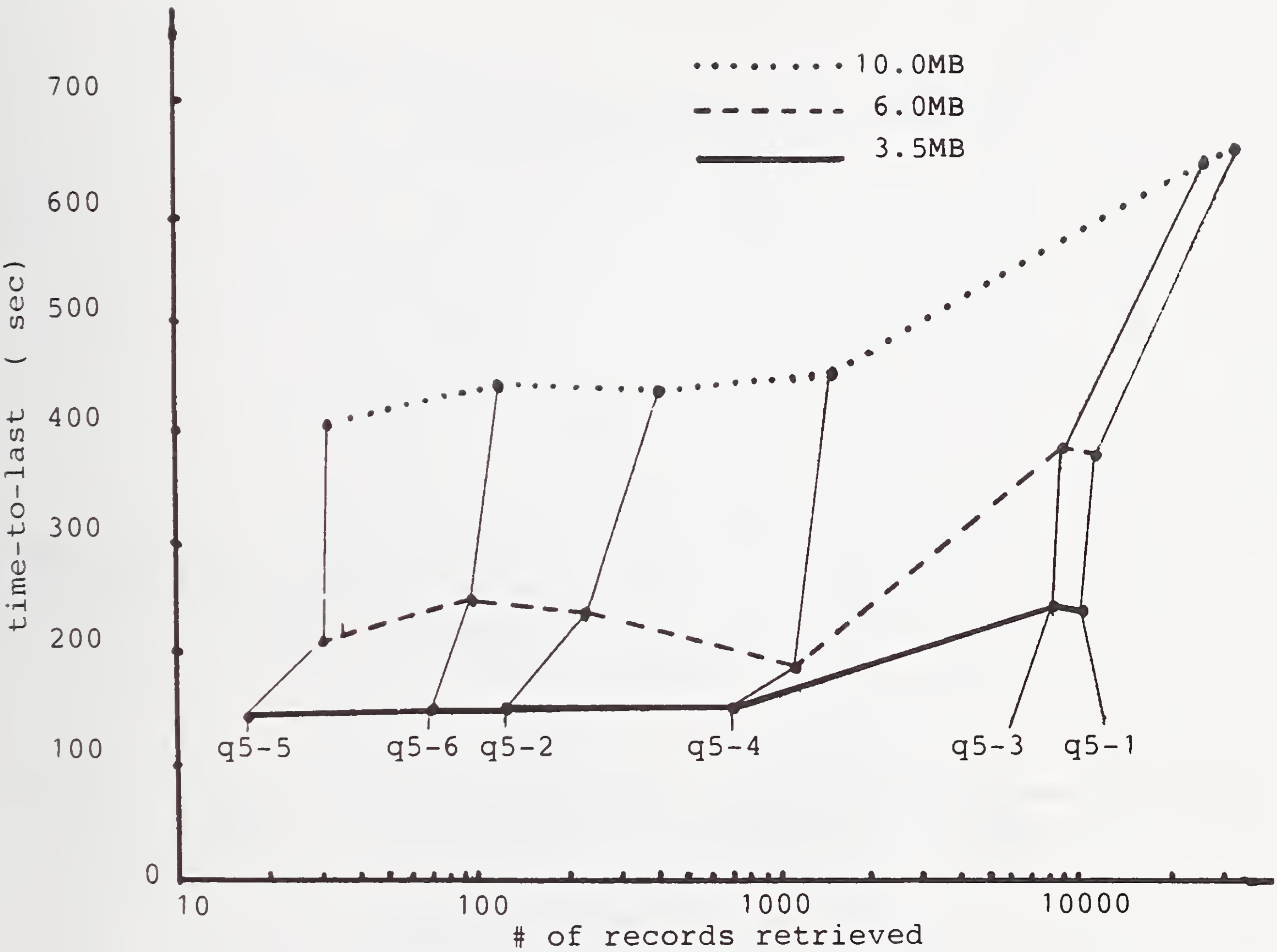


Figure 7.3: Records Retrieved vs. Time to Last Record.

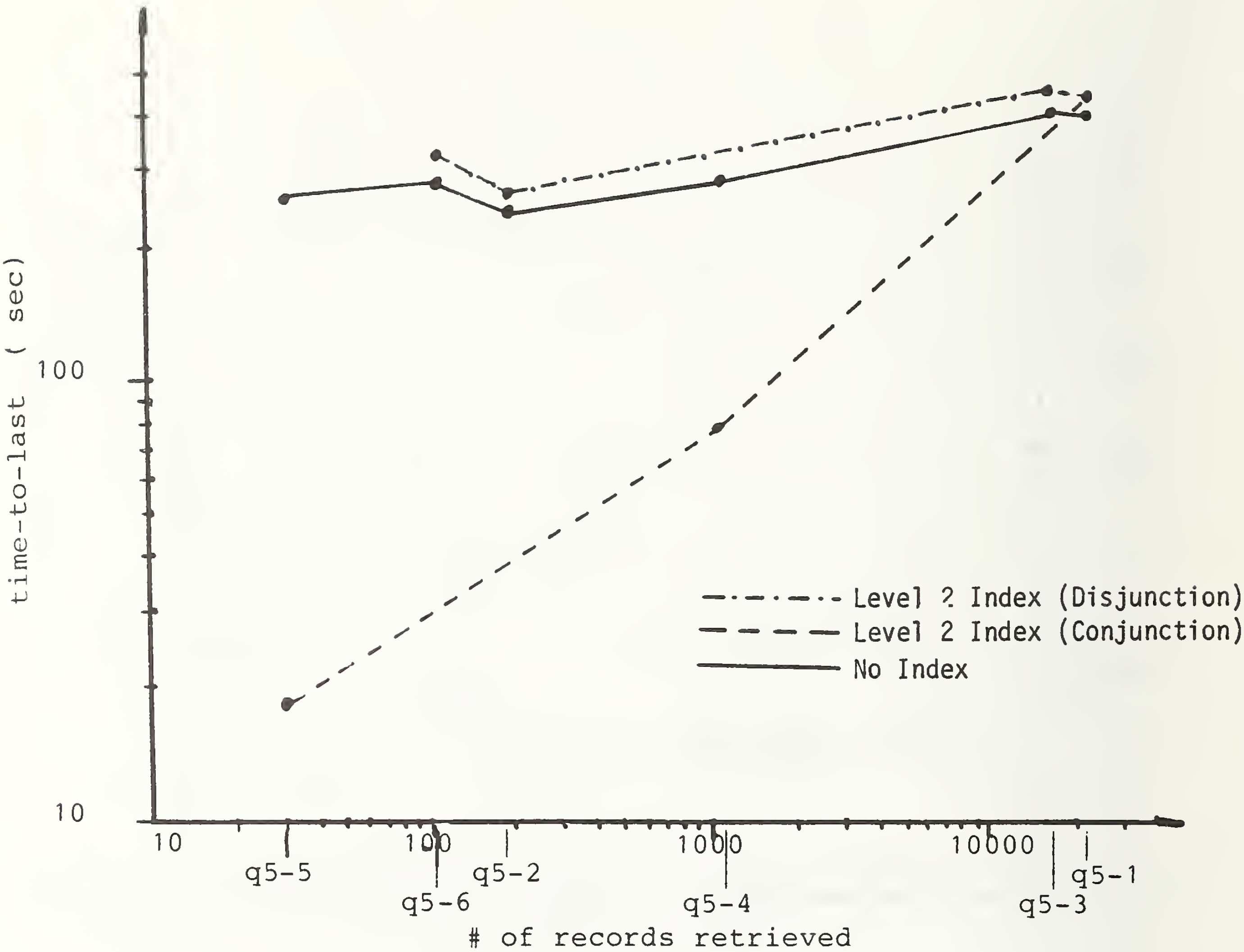


Figure 7.4: Single Relation Index Tests (6 MB Database).

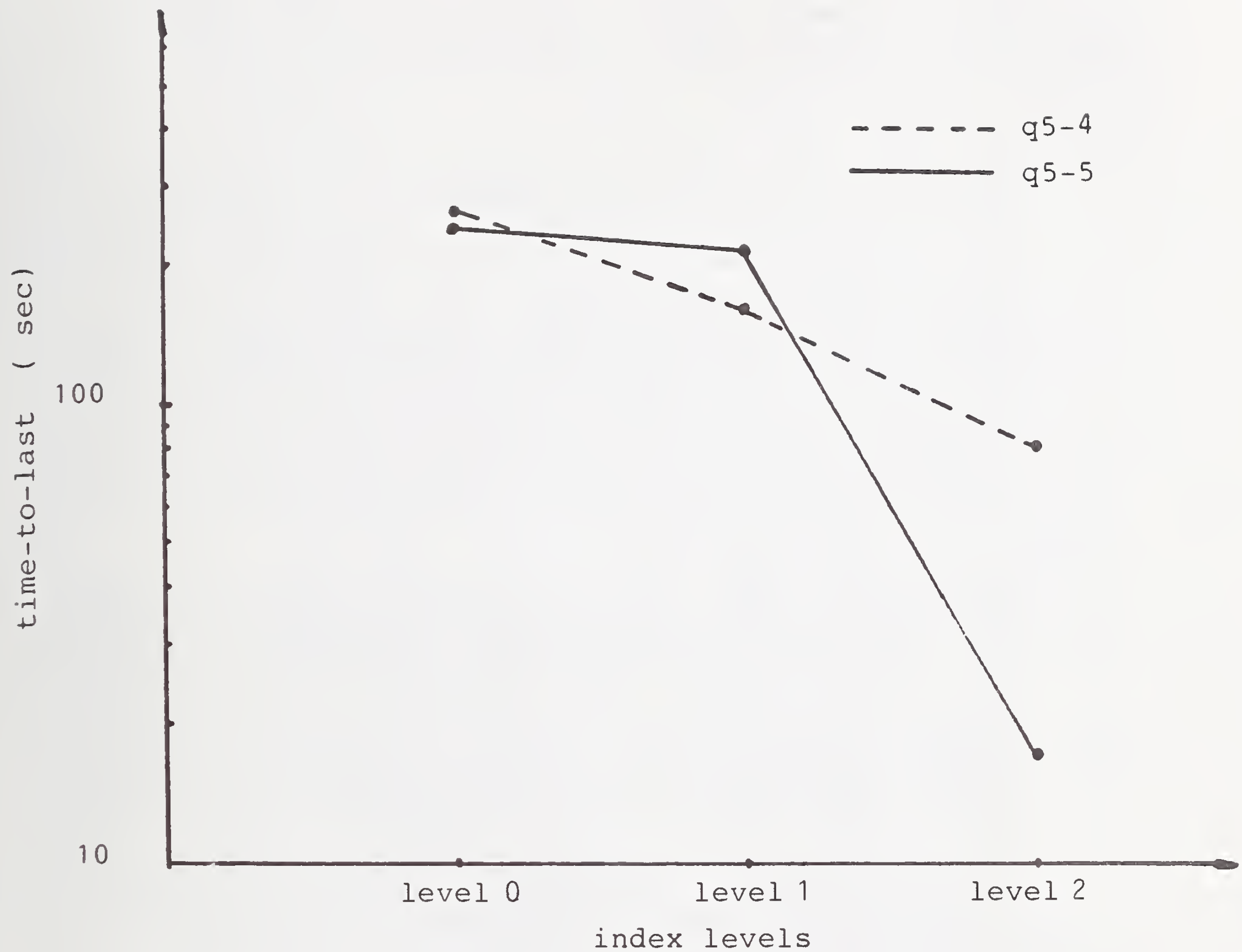


Figure 7.5: Index Levels vs. Time to Last Tuple (6 MB Database).

adding another 'OR' clause in query q5.6. Although the new condition used the indexed attribute ACAD_DISC, the response time for the result with level 2 indexing was similar to the response times for no indexing and level 1 indexing.

The performance of query execution did not always benefit from the use of indexes. This result was found to be valid over all database system architectures. The benefit of indexes was influenced by the following parameters:

1. **Hit Ratio.** The hit ratio refers to the percentage of records retrieved from a relation. Indexes are useful for retrieving a relatively low percentage of the file because they can avoid the searching of the entire relation sequentially. To retrieve a large amount of data, however, the extra index accesses become a burden and sequential searching could be more efficient. The time-to-last performance in Tables MINI.3 and MINI.4 showed that the high hit ratio penalized the performance of queries 1-2, 1-3, 1-4, 1-5 and 1-6. The hit ratios of these queries were determined from Table MINI.1 as 55.8%, 56.5%, 38.7%, 13.9% and 21.7% respectively.

2. **Disjunctive Index.** The use of a disjunctive condition (terms connected by 'OR') in a query can sometimes make index processing more difficult. It is well known that if one of the terms in the disjunction is not indexed, then any other index in the condition is not useful. However, if all the terms are indexed, or if two terms in a disjunction use the same index, then the performance depends on the particular implementation. Many database systems are not able to process the 'OR' index efficiently. In these benchmarks, performance penalities for disjunctive indexing can be observed, for example, on queries 2-3, 2-4 and 2-5.

3. **Range Index.** The use of index to solve a range query may have a similar effect as the disjunctive index. This effect can be observed on queries 3-3, 3-4, 3-5, 3-6, and 5-6.

The size of the database had a significant effect on performance for the tests when no indexes were used. No single relation queries would complete under 30 minutes for the 10 MB database.

"Effect of Buffering"

In order to determine if there was any buffering effect among queries accessing the same set of relations, two different mixes of queries were run. The first set of query mix ran similar queries in sequence. The response time could be reduced if the system was able to take advantage of the data kept in the buffer from previous accesses. The data for these runs are found in Table MINI.5. The other query mix ran queries in random order (Table MINI.2). Figure 7.6 shows that the time-to-last for query set 5 was not reduced significantly by running similar queries for any number of records retrieved. The same observation was valid for all query sets, for all database sizes tested, and for all database system architectures.

"Effect of Sorting"

Next the effect of sorting on the response time was examined. Table MINI.6 contains the results of the sorting tests. Figure 7.7 demonstrates the significant increase in response time when query results must be sorted. The comparison is shown for query set 1. One set of data required sorting the resulting records, one did not. The response time to the last record retrieved was significantly longer for the set which required sorting, regardless of the number of records retrieved. The difference between the response times to last record retrieved also increased with the number of records retrieved. Since sorting involves a non-polynomial complexity, this result was expected. A similar result was seen in each database system architecture tested.

A test was also run to see if sorting performance was improved by adding level 2 indexes (Table MINI.7). No significant improvement over level 1 indexing was found.

"Effect of Aggregate Functions"

The inclusion of aggregate functions in query sets 4 and 5 was tested. The number of result records are shown in Table MINI.8 and the response times are shown in Table MINI.9. In Figure 7.8 the time-to-last of query set 5 with and without the aggregate function are compared. The performance difference was not important until the result sizes increased beyond 10,000 records. Then the cost of performing the aggregate function became significant. Table MINI.10 displays the response time values for aggregate queries with level 2 indexing. Here significant decreases were found in response times for queries wherein the aggregations were based on groupings by indexed attributes. In queries qx4.3, qx5.4, and qx5.5, the time-to-last values were much smaller when the attribute to be grouped for aggregation was indexed.

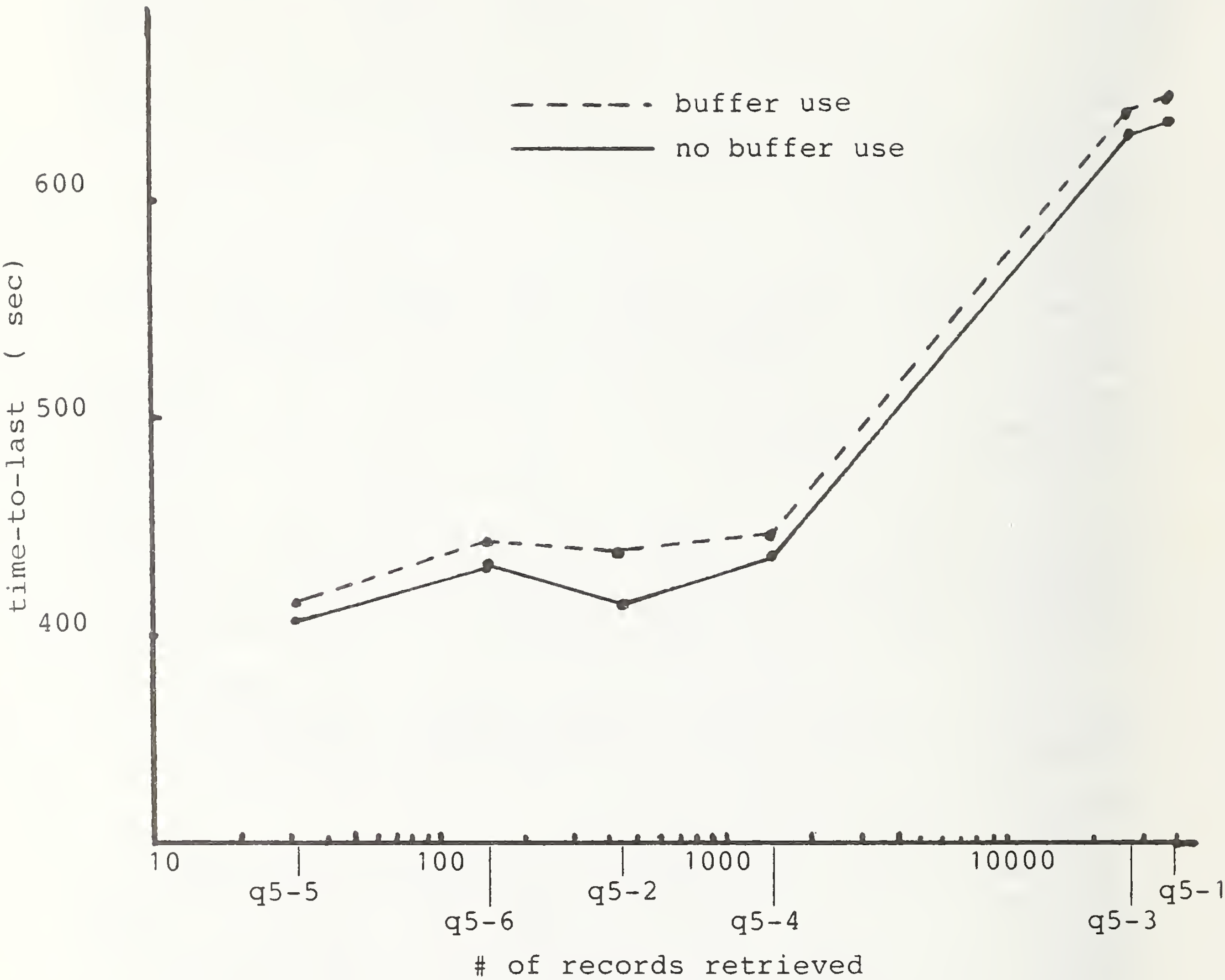


Figure 7.6: Effect of Buffering (10 MB Database)

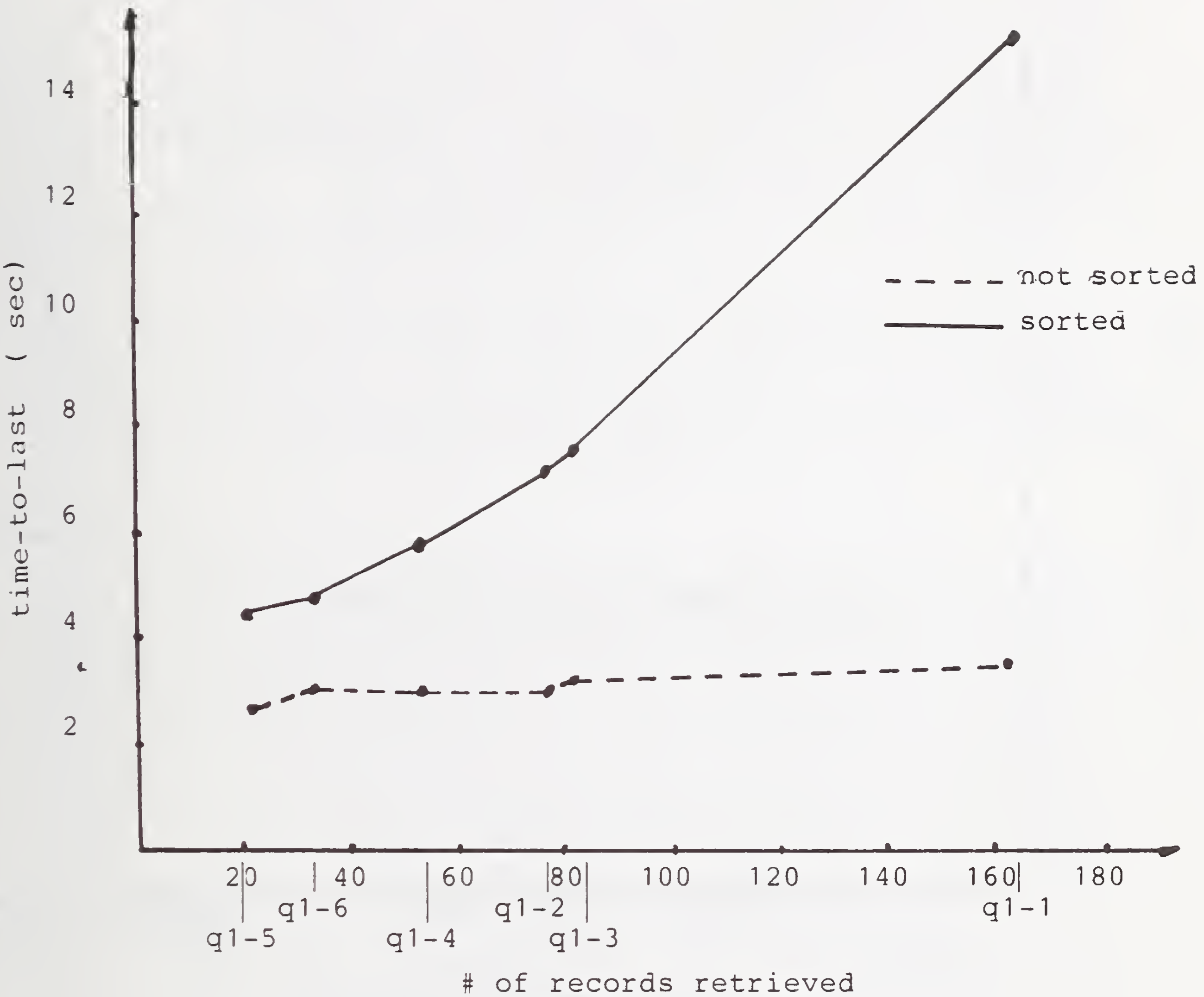


Figure 7.7: Effect of Sorting (10 MB Database)

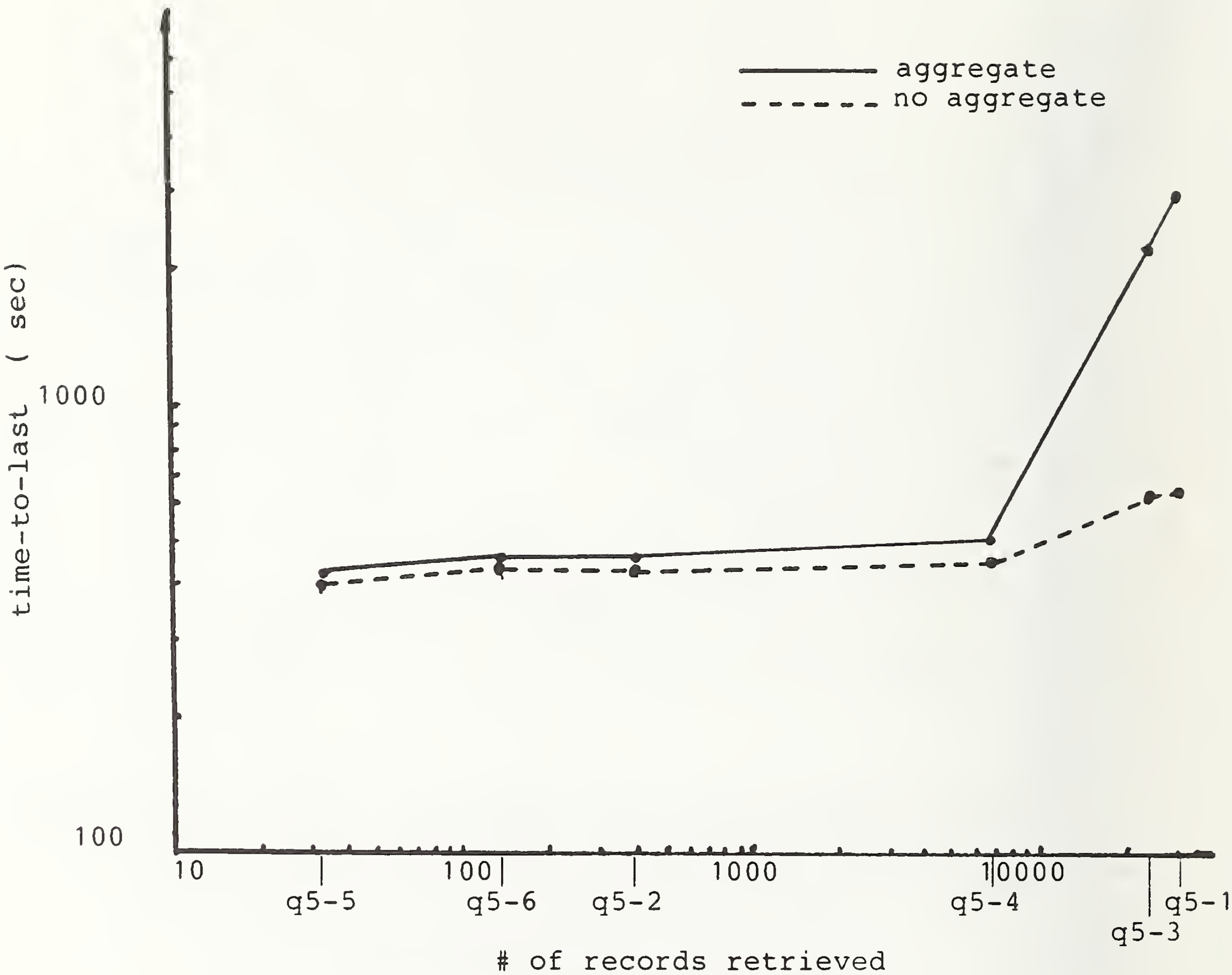


Figure 7.8: Effect of Aggregation (10 MB Database)

7.2.2 Multiple Relation Queries.

"Response Time vs. Query Complexity"

The multiple relation result sizes are given in Table MINI.11. Similar to the single relation queries, within each query set, the number of records retrieved reflected the complexity and makeup of the query conditions. The benchmark results provided several clear observations. Query set 10, the four relation query, did not execute under any conditions. Multiple relation queries would only run with level 1 and level 2 indexes. Thus, at a minimum, unique indexes on relational keys must be provided for satisfactory join performance.

Table MINI.12 contains the performance data for the multiple relation queries with level 1 indexing. Figure 7.9 shows the effect of the increasing database size on the time-to-last performance of queries q8.1, q8.2, and q8.4. The behavior was approximately linear on a log scale. Other test queries had similar performance behavior.

"Response Time vs. Indexing"

The effect of indexing on multiple relation queries provided a similar observation as in the single relation case. Secondary key indexes were valuable when they could be used effectively in the query predicate. The performance data for level 2 indexing is found in Table MINI.13.

Consider the time-to-last data in the following Table.

Table 7.1: Multiple Relation Index Effect (10 MB Database)

Response Time is Milliseconds		
Query	Level 1 Indexes	Level 2 Indexes
q6-2	654,420	399,890
q6-3	1,358,270	1,344,760
q7-2	2,973,060	457,830
q7-3	2,975,690	2,963,660
q8-2	7,729,520	223,200
q8-3	7,837,480	8,017,500

For query sets 6, 7, and 8 the second query (e.g., q7-2) added an equality selection condition on an attribute that was indexed under level 2 indexing. The improvement in response time was dramatic when an index on the selection attribute was placed on the database as shown in the above

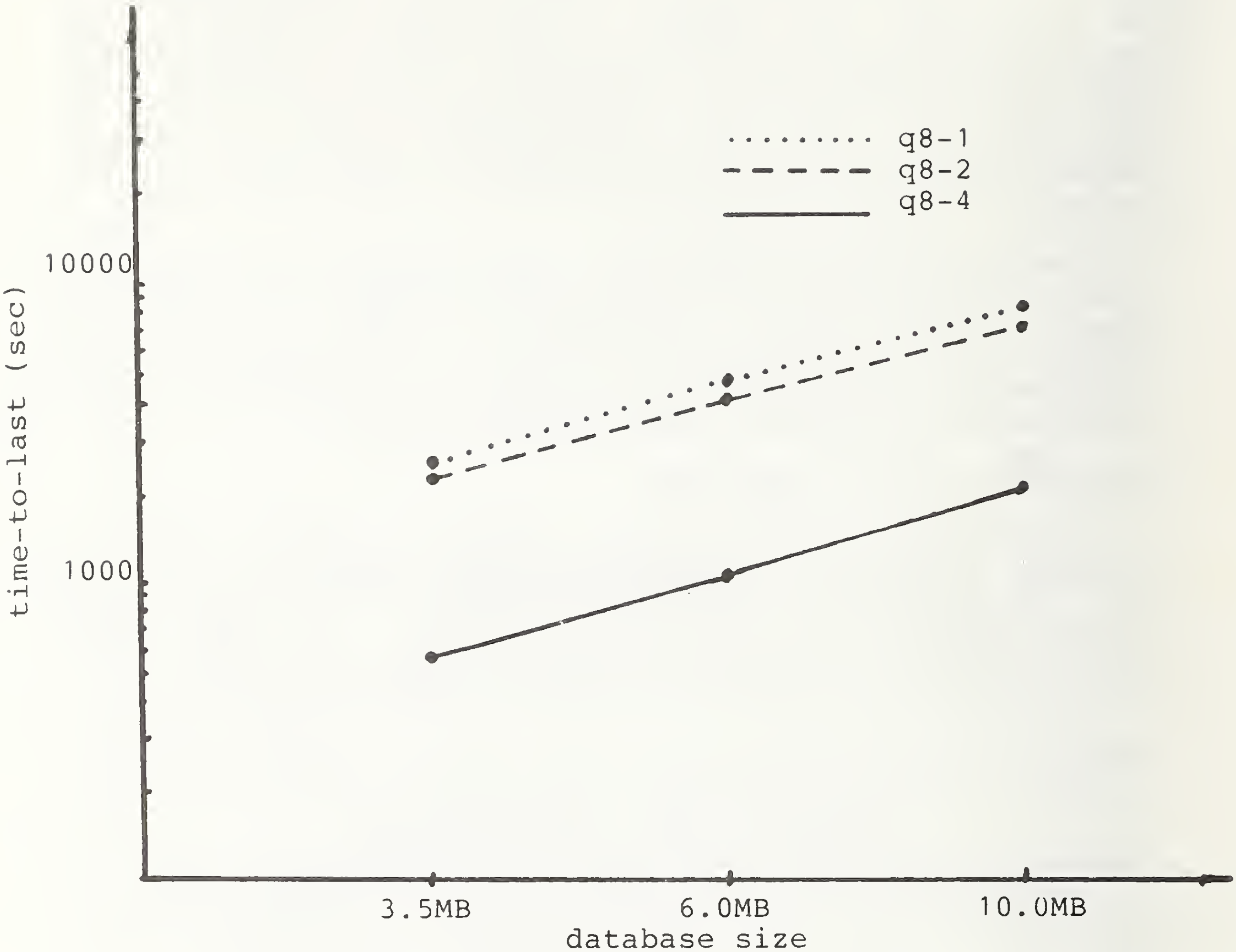


Figure 7.9: Database Size vs. Time to Last Record

data. The third query in these query sets added another equality selection condition to the predicate connected by an 'OR' operation. For example, query q7.3 is:

```
q7-3:  select pers_data.ssn, citizen, vet_pref
         from pers_data, pers_misc
         where pers_data.ssn = pers_misc.ssn
         and handicap = '01' or handicap = '49';
```

The level 2 indexing, however, provided no performance improvement over the level 1 indexes, even though the OR'ed attributes were indexed. These examples provide clear evidence for the discussion in the previous section that indexing is not always beneficial in a minicomputer database. Here the presence of the 'OR' operation did not lead to efficient index use on the queries.

"Special Studies"

The following observations are based upon the response time data on the special case multiple relation queries.

1. **Order of query condition.** The rearrangement of clauses in the query condition resulted in an improvement in response time. On the 3.5 MB database query scl-1 had a time-to-last value of 1330 milliseconds while query scl-2 ran in 630 milliseconds. On the 6 MB database query scl-1 ran in 1250 milliseconds and scl-2 ran in 680 milliseconds. Similarly, on the 10 MB database scl-1 ran in 1410 milliseconds and scl-2 ran in 660 milliseconds. The tested database system did not seem to recognize when identical clauses in a query were rearranged. Different access strategies seemed to be produced.

2. **Implicit vs. explicit join.** The implicit vs. explicit join queries were run on the 3 different database sizes. The resulting data are presented here:

Table 7.2: Response Time Data for Special Case 2

Response Time in Milliseconds						
Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
sc2-1	1,460	1,420	1,410	1,490	1,420	1,650
sc2-2	145,560	148,670	147,730	283,090	547,050	902,920

As the data shows, the system performed very poorly on the explicit query. As in the first special case, the minicomputer database system did not appear to recognize certain types of identical queries. This performance was not attributed to the system architecture, however.

3. **Join optimization.** The join optimization test was run on the three database sizes. On the 3.5 MB database the result values for time-to-last were:

   Database System Optimization
   379,260 ms.

   Simulated Sort/Merge
   193,290 ms.

   On the 6 MB database the result values for time-to-last were:

   Database System Optimization
   839,140 ms.

   Simulated Sort/Merge
   365,910 ms.

   On the 10 MB database the result values for time-to-last were:

   Database System Optimization
   1,388,450 ms.

   Simulated Sort/Merge
   662,320 ms.

It was observed that the tested database system query strategy resulted in response times that were nearly double the response times of the sort-merge strategy that was simulated with the other 3 queries. It can be concluded that this particular database system did not take full advantage of the sort-merge join optimization techniques that were beneficial for many queries. Again, however, this result cannot be attributed to an architecture weakness, but to the particular database system implementation.

7.2.3 Updates.

The performance results for the updates are found in Table MINI.14. Since no records were retrieved, only one response time figure is presented. The following observations are made.

1. The updates would only run with indexes. Only response times for level 1 and level 2 indexes are given.

2. The extra overhead of updating additional indexes is clearly seen in queries qI-2 and qD-3 on the EDUCATION relation. In these two queries, three level 2 indexes had to be updated when a record was inserted into or deleted from the relation. With level 1 indexes, only one index had to be updated. The conclusion is that additional indexes may significantly impact performance during updating in a minicomputer database system.

7.2.4 Multiple User Results.

Multiple user tests with 1, 2, and 3 users were run. A random job script with 5 single relation queries and 4 multiple relation queries was selected. The job script consisted of queries {q1-1, q2-2, q3-3, q4-4, q5-5, q6-5, q7-7, q8-3, q9-2}. The runs were made on the 6 MB database with level 1 indexes. The job script was varied in three ways to test the effect of potential conflict and data locking. In order 1, the job scripts for each user were identical and in the same order. In order 2, one of the job scripts was reversed. In order 3, the job scripts for each user were scrambled. The response times are recorded for the completion of the job scripts of queries. The times in the table below are in seconds.

Table 7.3: Performance Data for Multiuser Tests (6 MB Database)

Response Time in Seconds			
# of users	Order 1	Order 2	Order 3
1	8,435.95	8,339.69	8,511.58
2	15,696.23	15,029.96	15,673.71
3	23,626.54	23,639.76	23,631.29

This data is graphed in Figure 7.10. The following observations are made:

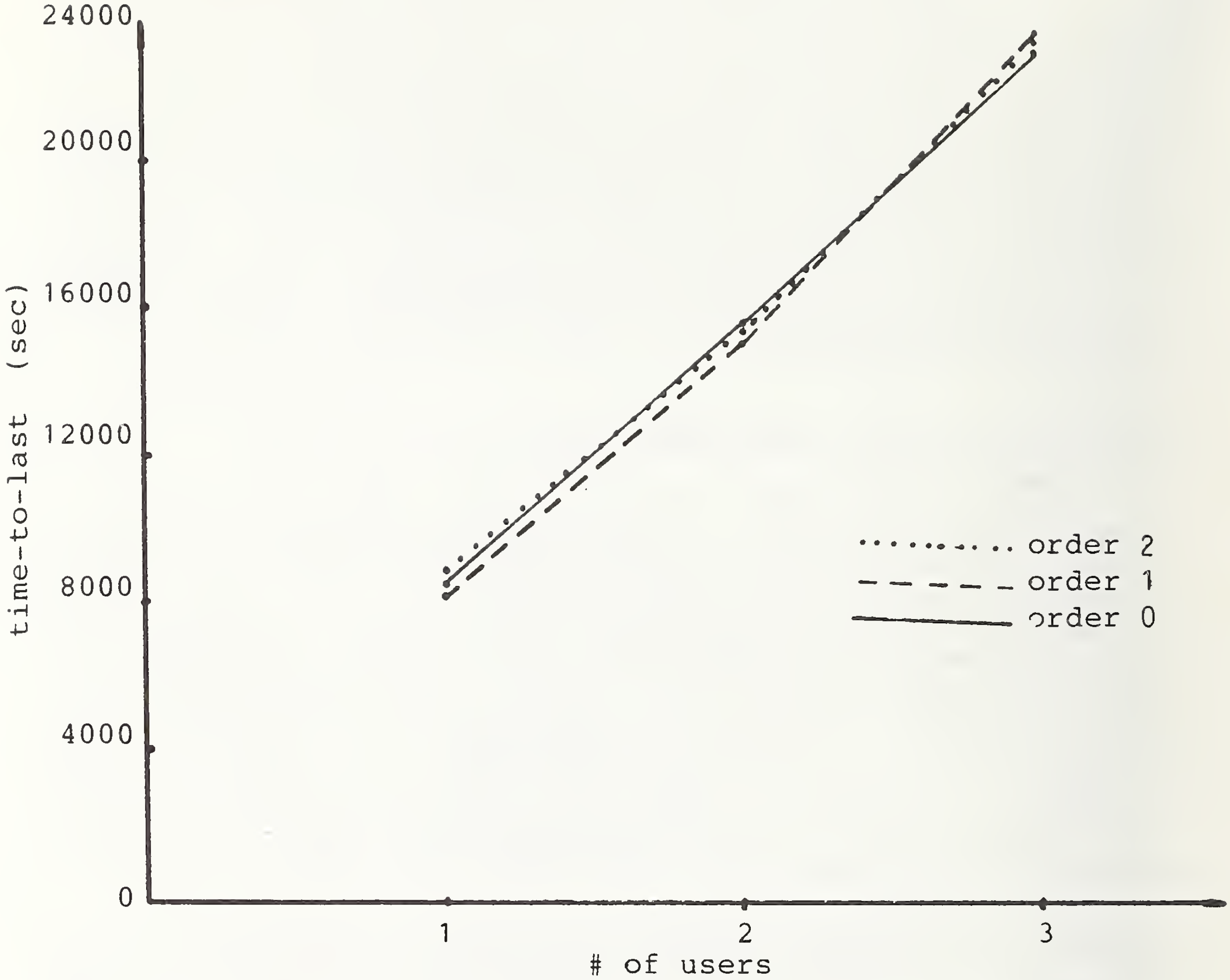


Figure 7.10: Multi-User Results (6 MB Database)

1. Rearranging the order of the queries in the job script had no effect on the response times.

2. The graph clearly shows the linear increase of the response times over the range of users in the study.

7.2.5 Background Load Results.

To test the effect of a background load on database performance, a sort routine was designed and programmed on a medium size data file. This job was estimated to be 75% CPU intensive and 25% I/O intensive. Running alone on a dedicated VAX 11/750 the sort routine had an average response time of 1,106.29 seconds.

Two job scripts were built for this benchmark. Job script 1 contained all of the single relation query sets (1 - 5). Job script 2 contained the multiple relation query sets 6, 7, and 9. Each job script was run with 0, 1, 2, and 3 background jobs and the response times of the job scripts and the background jobs was measured. The benchmarks were run on the 6 MB database with level 1 indexes. As with all of the benchmarks in this report, the costs of the ´runner´ program was considered as negligible.

The performance effects of running a background load was measured in two ways. First the effect of the background jobs on the database job scripts was studied. The effect of the database load on the response times of the sort job was also studied.

The table below lists the response times of the two job scripts as the background load was increased from 0 to 3 jobs.

Table 7.4: Performance Data for Job Scripts (6 MB Database)

Response Time in Seconds		
# of background jobs	single relation job script	multiple relation job script
0	6,369.68	17,110.22
1	7,353.20	19,231.50
2	8,428.52	20,426.44
3	9,351.03	21,592.07

Figure 7.11 shows the linear effect that the addition of more background jobs had on the database job scripts. This result was expected in a minicomputer architecture.

For the reverse benchmark, the average response times for the background sort jobs were:

Table 7.5: Performance Data for Background Jobs

Response Time in Seconds		
# of background jobs	single relation job script	multiple relation job script
1	4,243.75	2,983.56
2	6,024.80	4,223.21
3	7,839.60	5,352.08

The response times of the background jobs increased linearly as the number of background jobs was increased. The most interesting observation was that the sort job ran significantly slower while the single relation job script was running than when the multiple relation job script was running. This was true even though the multiple relation job script required nearly 2.5 times more process time to complete. This was due to the number of individual queries in the single relation job script (30 vs. 18 in the multiple relation job script). The CPU contention caused by activating, parsing, and optimizing each single relation query affected the sort programs to a greater extent than the handling of the fewer multiple relation queries.

7.3 Summary

The following points summarize the major results of the benchmark study of the minicomputer database system architecture. These points can be used as guidelines for evaluating the performance of minicomputer database systems. The first 6 points show results of similar nature as in the microcomputer database benchmark. A comparison of performance over the three tested architectures is presented in Section 9.

1. In general, query complexity was inversely proportional to the number of records retrieved. Time-to-first-record was greater when fewer records were retrieved, since the system took longer to find the

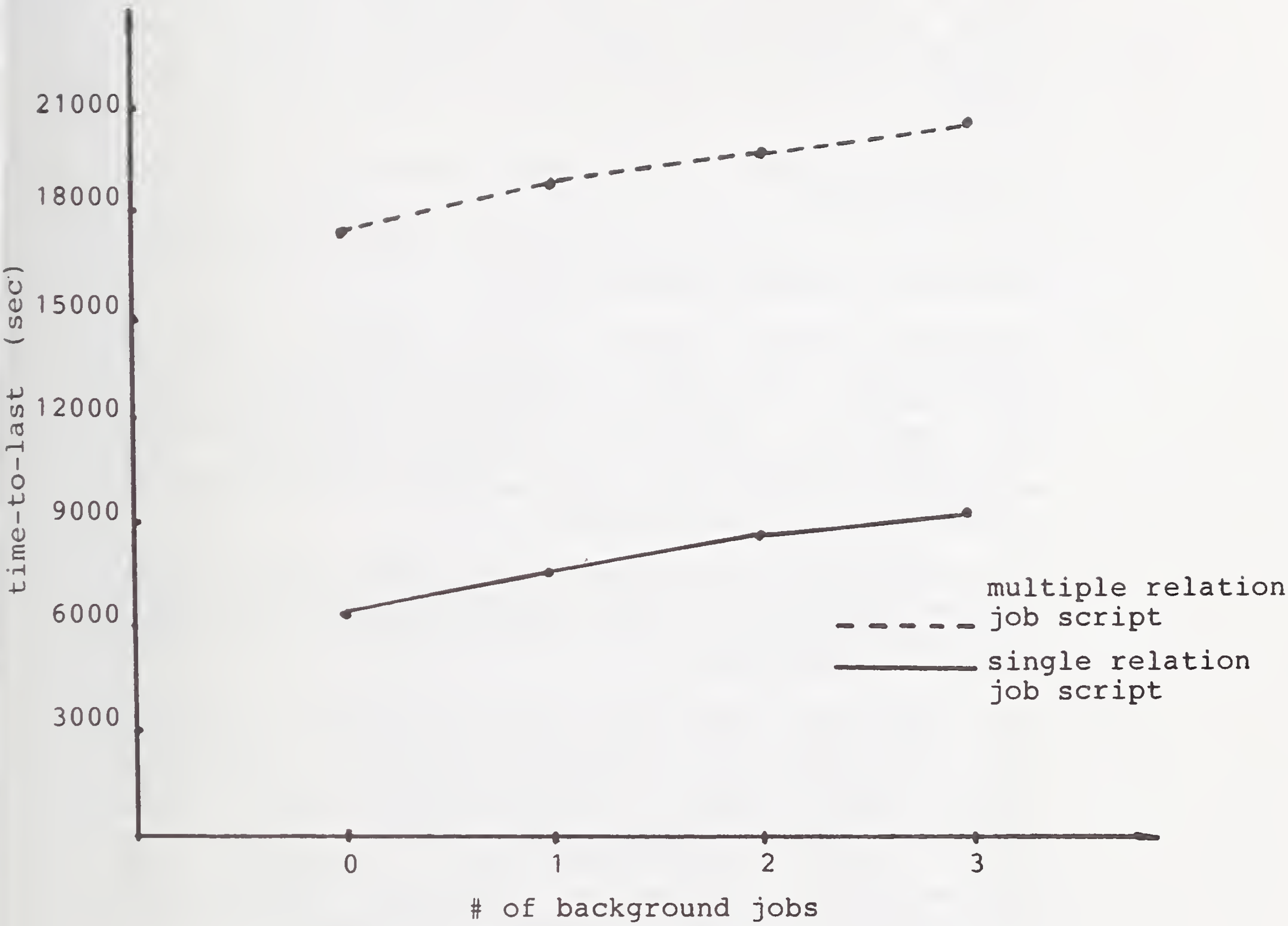


Figure 7.11 Background Load Results (6 MB Database)

first record. Time-to-last-record was greater when a larger number of records were retrieved, due to the time required to access and transmit additional records.

2. Indexing reduced response time if the index could be used effectively in the query access strategy. Several types of queries were identified that did not allow the index to be used effectively. These cases included queries with high hit ratios, queries with disjunctive conditions, and queries with range conditions.

3. The order in which queries were run had no influence on performance.

4. Sorting and aggregation functions added significant overhead to query response time.

5. Multiple relation queries (join queries) required considerably more time to complete than single relation queries. Queries involving four relations would not complete. Multiple relation queries would not run on databases of any size without indexes. For acceptable performance, indexes had to be present on primary keys in the database.

6. The arrangement of conditions in a query had an effect on performance. This was attributed to the particular database system's implementation, not the system's architecture.

7. Join conditions should be clearly written to identify the attributes involved in a matching operation (e.g., pers_data.ssn = education.ssn). A significantly poorer response time resulted when the same query was tested and each join condition was made explicit (e.g., (pers_data.ssn = 300378541) AND (education.ssn = 300378541)). This was attributed to the particular database system's implementation, not the system's architecture.

8. Update performance was significantly improved by having clustered indexes on the primary keys in a relation.

9. The presence of multiple users had a linear effect on the performance of the job scripts in the database system.

10. Adding background load jobs increased the response times of job scripts linearly based upon the number of background jobs on the host computer system. A reverse benchmark showed that the performance of the background jobs is affected by the number of database queries entered into the database system through the host computer system.

8. DATABASE MACHINE BENCHMARK

8.1 Benchmark Implementation and Execution

A comprehensive discussion of the system configuration, implementation, and benchmark execution for the representative database machine architecture is given in Appendix D.

8.2 Benchmark Analysis

In this section selected analyses are presented based upon the performance results that were obtained from executing a benchmark on the database machine. The discussion of results is divided into five sections: single relation queries, multiple relation queries, update queries, multiple user results, and background load results. The resulting performance data from the tests on the database machine are found in Appendix C.3. There is no claim to present here a critical analysis of the performance of any commercial database machine.

8.2.1 Single Relation Queries.

One of the most important measures for a database system's performance is the query response time. In most on-line applications, response time is defined as the time from the submission of the query until the query result is available. Two different timings are important: the time until the first result record is obtained and the time until the last record is available. In this benchmark study, both of these timings are analyzed.

"Response Time vs. Query Complexity"

Figures 8.1 through 8.3 show the dependency of the response time on the query complexity and the amount of data retrieved. Table DBM.1 in the appendix contains the single relation query result sizes. The response times for the single relation queries with level 1 indexes are in Table DBM.2.

Figure 8.1 shows that the time-to-first decreases as the complexity of the queries decreases. This observation does not differ significantly for different database sizes of 3.5 MB, 6 MB and 10 MB. An intuitive explanation for this result is that when the query complexity decreases, more records are retrieved. If records are 'uniformly'

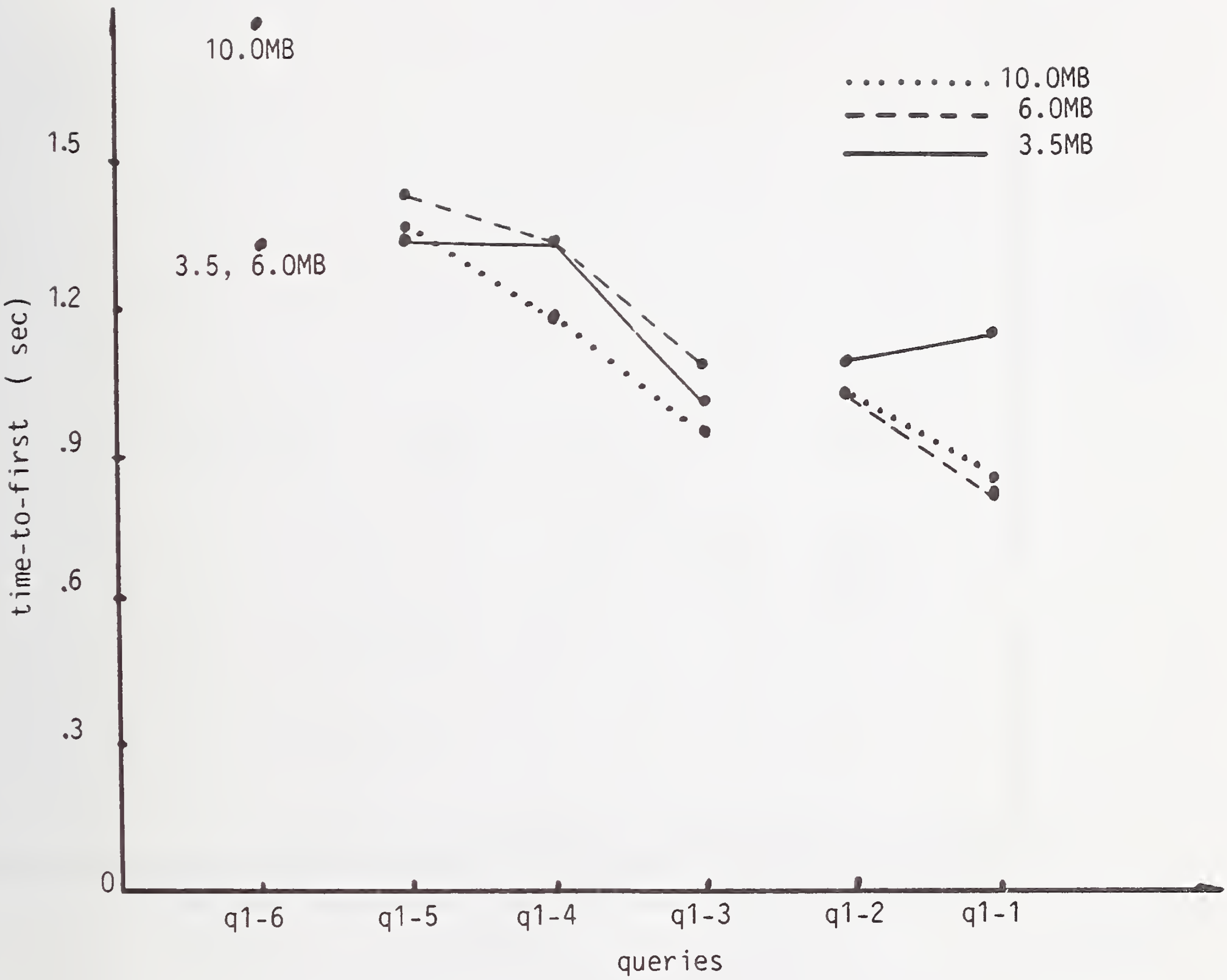


Figure 8.1: Query Complexity vs. Time to First Record.

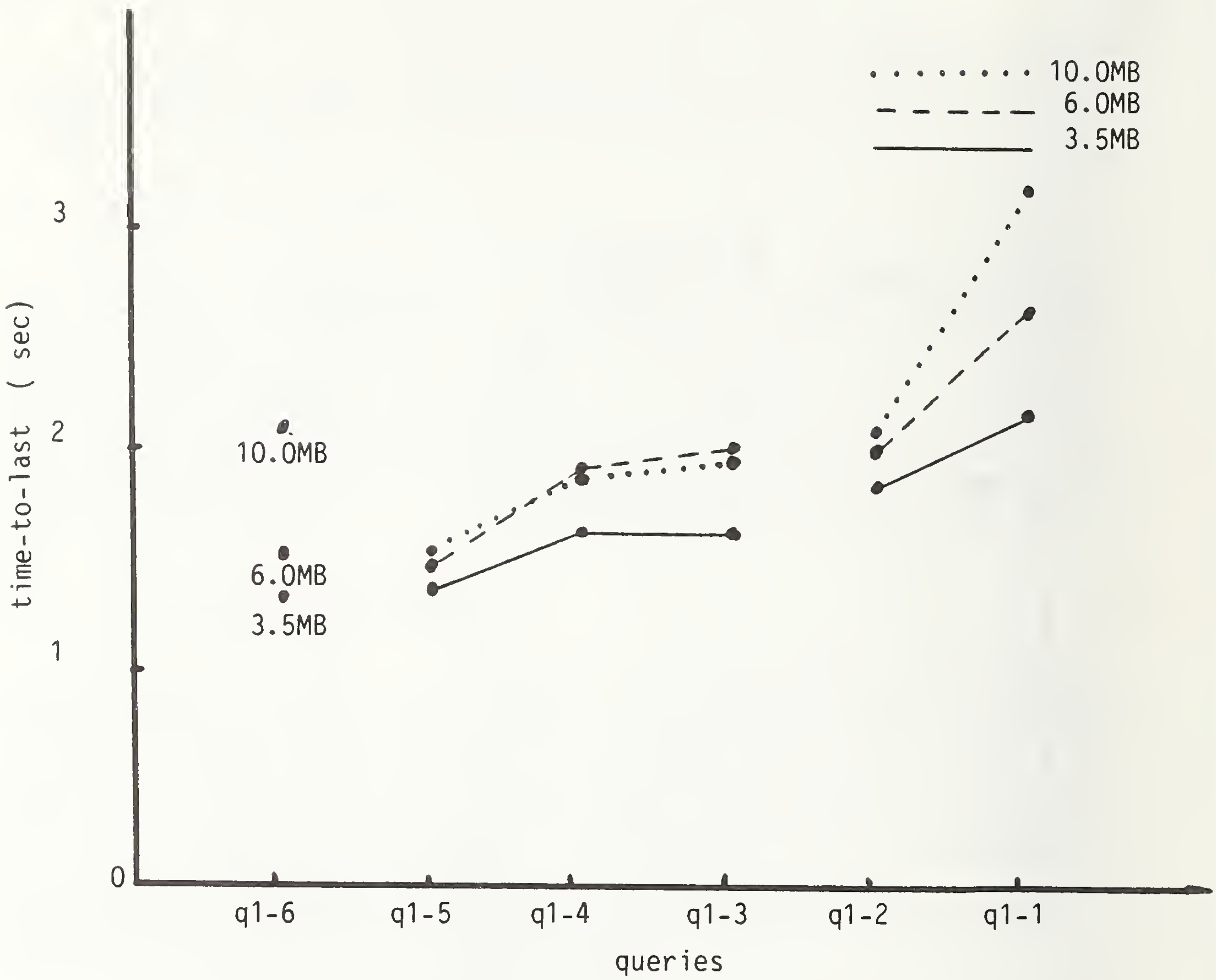


Figure 8.2: Query Complexity vs. Time to Last Record.

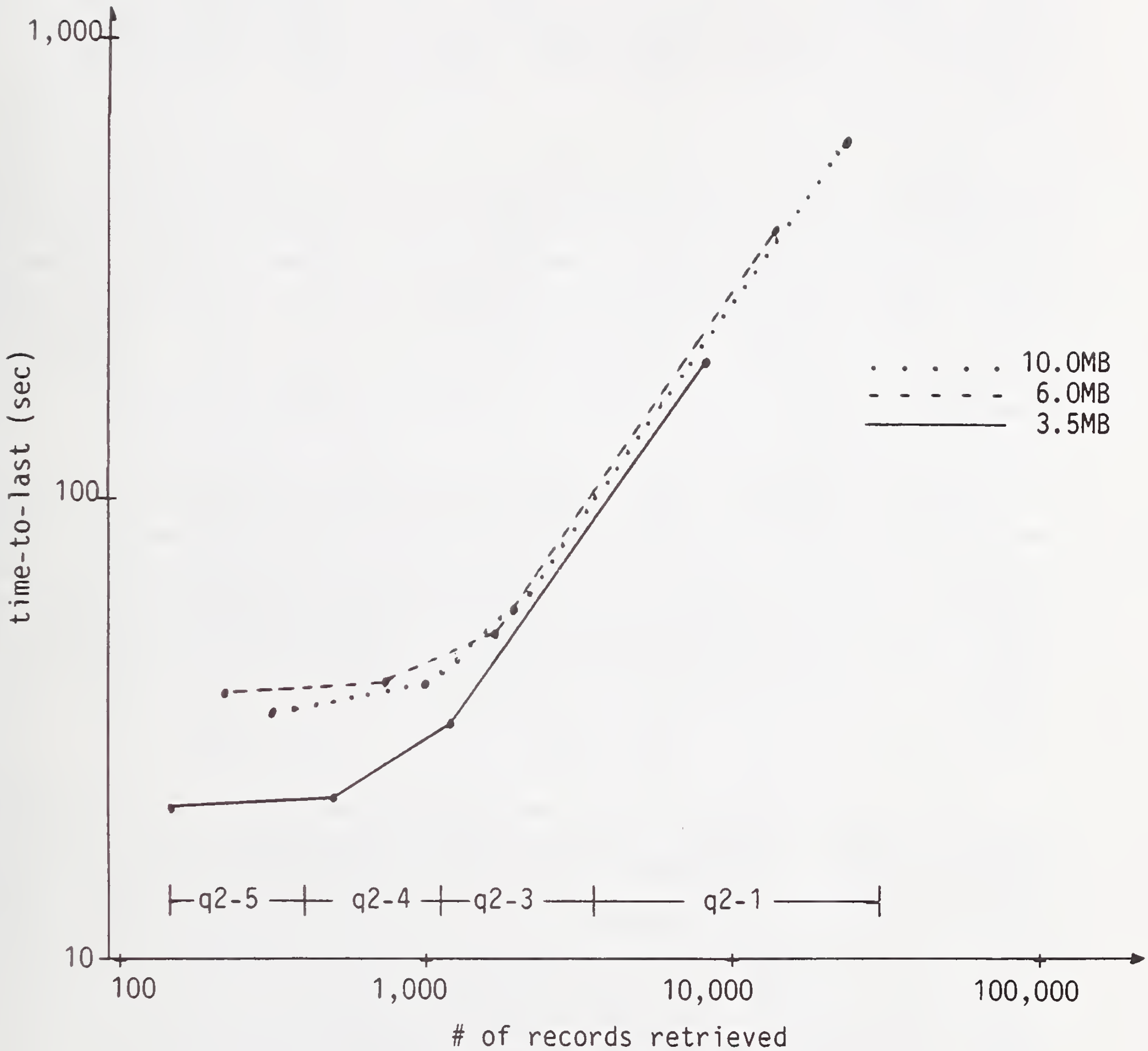


Figure 8.3: Records Retrieved vs. Time to Last Record.

distributed in the relation, the time for reaching the first record should be shorter. Obviously, this result will not be valid if indexes are used to process the query. The breaks in the curves correspond to the introduction of the 'OR' operator in the query condition. The addition of 'OR' had an effect of <u>increasing</u> the number of records retrieved. In order to isolate the 'AND' effect it was decided not to connect data points for the 'OR' operations in the figures.

On the other hand, Figure 8.2 shows that time-to-last increased as the complexity of the queries decreased. Here the factor which influenced response time to the last record was the total number of records retrieved which is inversely proportional to the query complexity. The curves show a sharp jump from query q1-2 to query q1-1 for all three database sizes. This is to be expected since for 6 MB, the number of records retrieved goes from 72 to 129 records and for 10 MB, from 78 to 163 records (Appendix C.3, Table DBM.1).

Figure 8.3 further reinforces this observation. When the time-to-last was plotted against the number of records retrieved, the curve showed an almost linear behavior. The initial rather 'flat' segment, where a smaller number of records was retrieved, was influenced by the query processing overhead including parsing, optimization, directory access, etc. Obviously, this result will not be valid if indexes are used to process the queries.

"Response Time vs. Indexing"

The effect of indexing was important for performance. Tables DBM.3 and DBM.4 contain the performance data for the single relation queries with level 2 indexes and no indexes, respectively. The indexes had different effects on time-to-first and time-to-last as the query complexity increased. When using indexes in queries q2-1 and q2-2, the time-to-first for the three different database sizes all increased as the complexity increased (Appendix C.3, Table DBM.3). On the other hand, the time-to-last increased as the complexity decreased (Appendix C.3, Table DBM.3-cont.). The major factor for time-to-first seemed to be the initial index search time. Once the index table was searched, the main factor that influenced the time-to-last performance was the time to fetch the qualified records. This result is summarized in the following Table 8.1.

Table 8.1: Response Times with Level 2 Indexes

Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
q2-1	867	766	750	189,733	359,150	599,700
q2-2	1,816	1,450	1,117	29,133	37,533	51,434

(Response Time in Milliseconds)

The performance of query execution did not always benefit from the use of indexes. This point was recognized for all of the tested database systems. The benefit of indexes is influenced by the following parameters:

1. **Hit Ratio.** The hit ratio refers to the percentage of records retrieved from a relation. Indexes are useful for retrieving a relatively low percentage of the file because they can avoid the searching of the entire relation sequentially. To retrieve a large amount of data, however, the extra index accesses become a burden and sequential searching could be more efficient. The time-to-last performance in Tables DBM.3 and DBM.4 showed that the high hit ratio penalized the performance of queries 1-2, 1-3, 1-4, 1-5 and 1-6. The hit ratios of these queries can be determined from Table DBM.1 as 55.8%, 56.5%, 38.7%, 13.9% and 21.7% respectively.

2. **Disjunctive Index.** The use of disjunctive condition (terms connected by ´OR´) in a query can sometimes make index processing more difficult. It is well known that if one of the terms in the disjunction is not indexed, then any other index in the condition is not useful. If all the terms are indexed or if the terms in a disjunction use the same index, then the performance depends on the particular implementation. In these benchmarks, a performance penality for disjunctive index was observed on queries 2-3, 2-4 and 2-5.

3. **Range Index.** The use of an index to solve a range query may have a similar effect as the disjunctive index. This effect was observed on queries 3-4, 3-5, 3-6, 5-4 and 5-6.

"Effect of Buffering"

In order to determine if there was any buffering effect among queries accessing the same set of relations, two different mixes of queries were run. The first set of query mix ran similar queries in sequence (Table DBM.5). The response time can be reduced if the system is able to take advantage of the data kept in the buffer from previous accesses. The second query mix ran queries in random order (Table DBM.2). Figure 8.4 shows that the response time to the last record retrieved was not reduced significantly by running similar queries for any number of records retrieved. The difference in response times ranges from 366 milliseconds for 3615 records to 217 milliseconds for 33,000 records - insignificant amounts. A similar result is observed on the tested minicomputer database system.

"Effect of Sorting"

Next the effect of sorting on the response time is examined. Table DBM.6 contains the performance data for the single relation queries with sorting. Figure 8.5 demonstrates the significant increase in response time when query results must be sorted. The comparison was made between similar query sets. One set required sorting the resulting records, the other did not. The response time to the last record retrieved was significantly longer for the set which required sorting, regardless of the number of records retrieved. For example, q2-1 requires 591,633 milliseconds, and q2-1 with a sorted result requires 1,080,216 milliseconds.

The difference between the response times to last record retrieved also increased with the number of records retrieved. For example, q2-5 and q2-5 with sorting retrieved 170 records and the difference between the response times was 7916 milliseconds. For q2-1 and q2-1 with sorting retrieved 33,000 records, and the difference was 488,583 milliseconds. Since sorting involves a non-polynomial complexity, this result was expected.

"Effect of Aggregate Functions"

Table DBM.7 contains the result sizes for the benchmarks run on queries with aggregate functions. Table DBM.8 contains the response time for the queries with level-1 index. The overhead of aggregation increased greatly as the number of records retrieved increased in the queries.

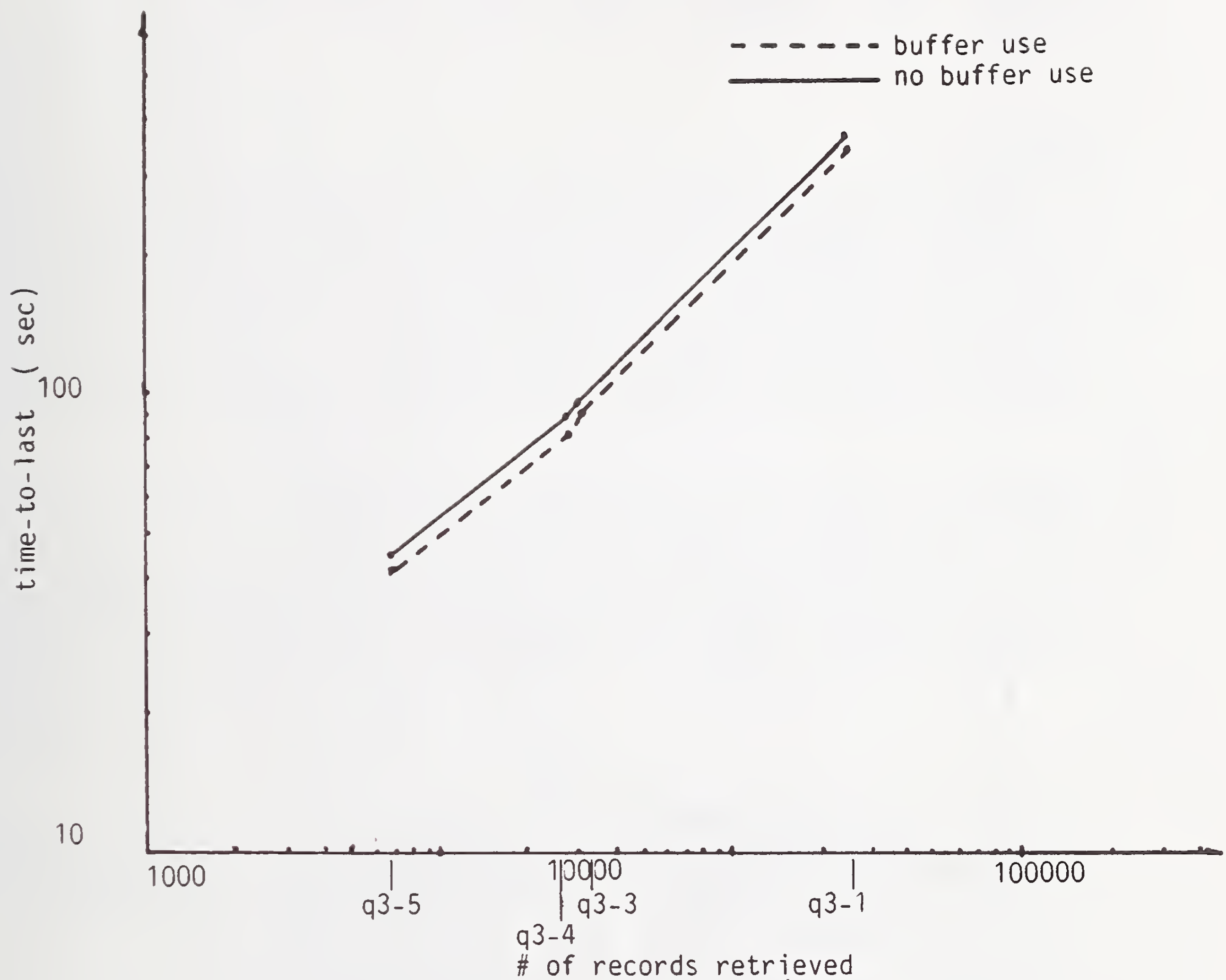


Figure 8.4: Effect of Buffering (10 MB Database).

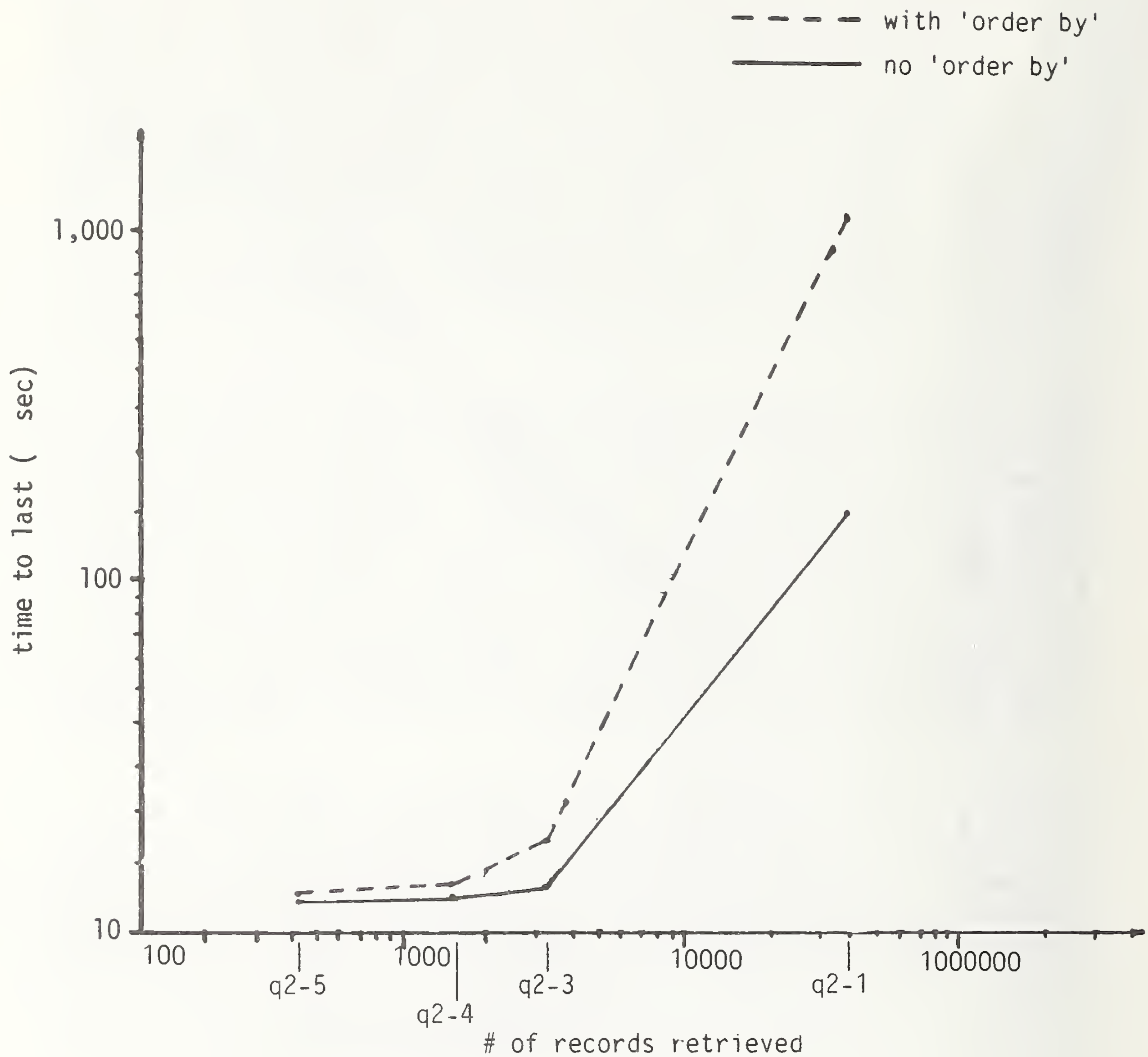


Figure 8.5: Effect of Sorting (10 MB Database)

8.2.2 Multiple Relation Queries.

Several benchmark tests were run using the multiple relation query sets 6 through 9. The result sizes of the queries is shown in Table DBM.9. Table DBM.10 holds the response times of the multiple relation queries with level 1 indexing. Tables DBM.11 and DBM.12 contain the response times of the queries with level 2 indexes and no indexes, respectively.

"Response Time vs. Indexing"

Of particular interest was the effect of indexing on the join performance. To process a join without using indexing, most systems use a nested loop technique. That is, for each record in the first relation, the entire second relation is accessed. The complexity for this type of algorithm is N**2 where N is the number of records (or pages) for each relation. Index support improves this algorithm by providing indexed access to the second relation and greatly reducing the complexity.

Figure 8.6 shows that the benefit of using clustered indexing was significant when the indexes were involved in the join. For the query set considered, the response times for queries using the index (level 2) were 3984, 3433, 4233 and 4984 milliseconds versus 107,834, 199,066, 693,250, and 617,567 milliseconds, respectively, when no indexes were used. The same beneficial effect of indexes was found in the minicomputer tests.

"Join Saturation Point"

Figure 8.7 shows that there was a significant jump for both time-to-first and time-to-last between database sizes 6 MB and 8 MB. This figure shows the data for query q6-1; however, all multiple relation queries have a similar behavior (see Table DBM.10). This could reflect the particular buffer management techniques being used by the system. Clearly, a saturation point was reached when the benchmark database approached 8 MB and the multiple relation join performance deteriorated after that point. Note that the saturation point of 8 MB in this case should not be interpreted as the capacity of the particular database machine. The saturation point will be dependent on the workload variables and machine configuration. While a similar saturation point in the minicomputer database system was not observed, this is not considered to be a result of the system architecture.

"Special Studies"

Figure 8.6: Effect of Indexing (6 MB Database).

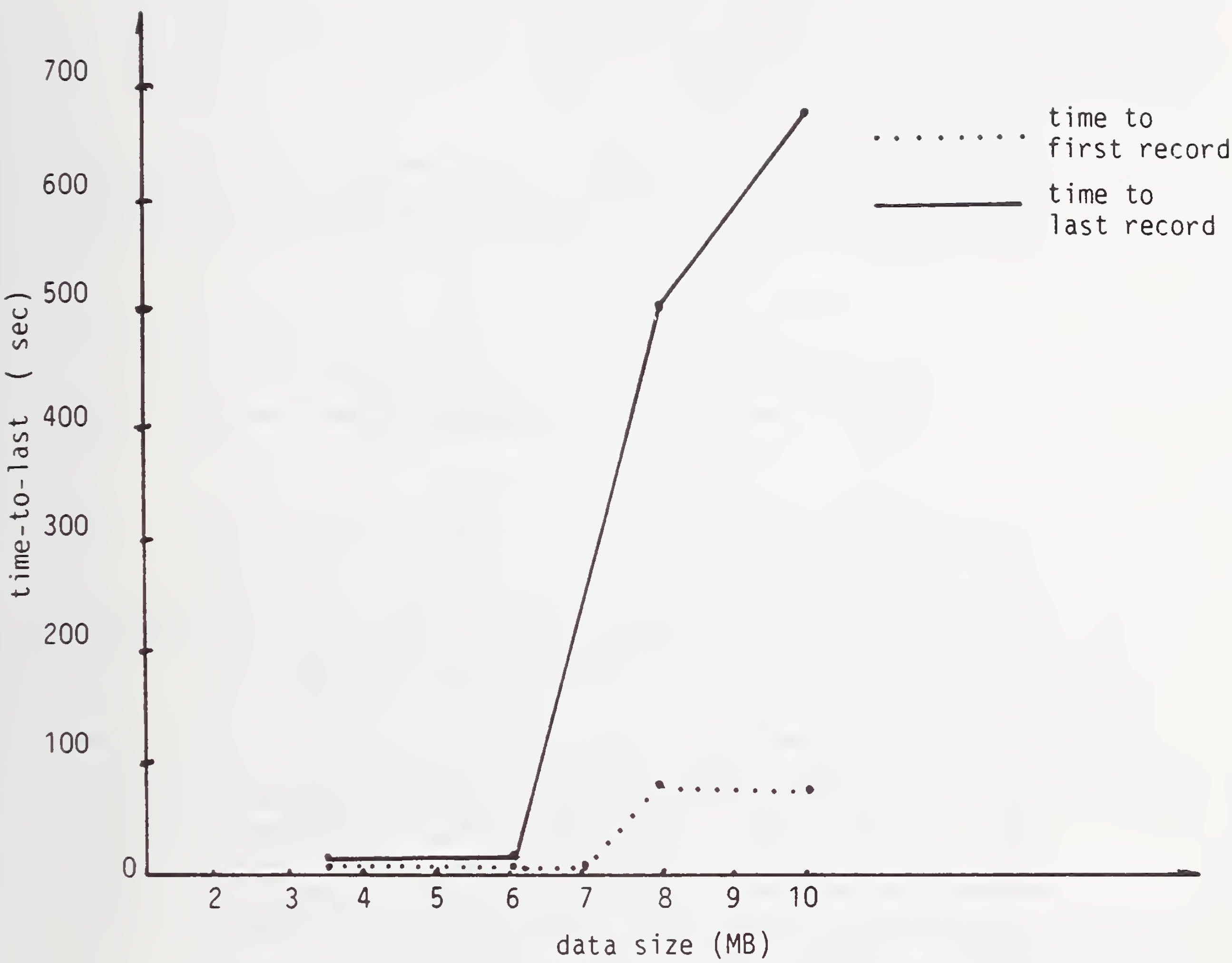


Figure 8.7: Database Size vs. Response Time
for Multiple Relation Query q6-1.

The following observations are based upon the response time data on the special case multiple relation queries.

1. **Order of query condition.** The rearrangement of clauses in the query condition resulted in no significant change in response time. On the 6 MB database query scl-1 ran in 1384 milliseconds and scl-2 ran in 1100 milliseconds. Similarly, on the 56 MB database scl-1 ran in 288067 milliseconds and scl-2 ran in 287584 milliseconds.

2. **Implicit vs. explicit join.** The implicit vs. explicit join queries (queries sc2-1 and sc2-2) were run on 3 different database sizes. The resulting performance was:

   Table 8.2: Special Case 2 Results

Response Time in Milliseconds			
Query	Database Size		
	3.5MB	6MB	10MB
sc2-1	1,250	1,400	50,700
sc2-2	1,333	1,017	51,600

   As can be clearly seen from the data, both queries performed identically. Thus, the database system recognized that both queries were equivalent.

3. The database machine performance on the first two special case tests differs significantly from the results on the minicomputer. The minicomputer database system did not recognize equivalent queries while the database machine system did. These differences are not attributed to the system architectures, however, but to the database system implementations.

4. **Join optimization.** The join optimization test provided some interesting results. The test was run on the 6 MB and the 56 MB databases. On the 6 MB database the results were:

   Database Machine Optimization
   198,017 ms.

   Simulated Sort/Merge
   128,483 ms.

   On the 56 MB database the results were:

Database Machine Optimization
1,990,567 ms.

Simulated Sort/Merge
1,488,299 ms.

It was observed that the database machine query strategy resulted in a higher response time than the simulated sort/merge strategy. On the 6 MB database the sort/merge strategy resulted in a 54% decrease in response time. On the 56 MB database the sort/merge strategy resulted in a 38% decrease in response time. Therefore, the database machine, similarly to the tested minicomputer database system, did not consider the sort-merge join optimization techniques that were beneficial for some join queries.

8.2.3 Updates.

The performance results for the updates are found in Table DBM.13 in Appendix C.3. The following observations are made.

1. The updates were executed on the 6 MB database with level 1 indexes and on the 10 MB database with no indexes. Little difference was observed based upon size and indexing, except for queries qD-3 and qM-2. The larger database with no indexes required a significantly greater time in these cases. By looking at the updates it was noted the condition in both queries contained a restriction on the SSN primary key. The absence of an index required a sequential search of the record to delete and modify. Thus, the increased response time.

2. The updates were run on the 56 MB database with all levels of indexing. Identical performance was observed except for updates qD-2 and qM-1 where the database with no indexing performed significantly poorer. Again, the basis of the poor performance can be attributed to the absence of the index on the CODE primary key in the LEGAL_AUTH table.

The conclusion drawn from both of these observations is that indexes on primary keys are helpful for satisfactory update performance in the database machine. The identical result holds for the other tested architectures.

8.2.4 Multiple User Results.

Multiple user tests were run with 2 users and with 3 users. A random job script with 5 single relation queries and 4 multiple relation queries was selected. In one test each user ran the script in the same order (order 1). In another test the order (order 2) of the scripts were scrambled. The performance results are shown in Figure 8.8. The following observations are made:

1. The presence of two users approximately doubled the response times of the job scripts over the single user times. The different orders had little effect.

2. The three user test showed a similar effect on performance. Response times approximately tripled over the single user times. The ordering of the queries in the different job scripts had an important impact, however. In the three user run with the job scripts in the same order, one of the users became ´locked´ and did not complete. This phenomenon was inexplicable.

The performance data for the multiple user tests on the database are presented in Table 8.3. The performance trends are similar to the minicomputer multiple user tests.

Table 8.3: Performance Data for Multiuser Tests (6 MB Database)

Response Time in Milliseconds		
# of Users	Order 1	Order 2
1	1,024,251	1,024,251
2	1,732,781	1,823,651
3	---	2,721,616

8.2.5 Background Load Results.

The purpose of the database machine architecture is to off-load database processing onto a separate processor. Only a small interface program remains in the front-end computer. In this way database processing will have little performance effect on the front-end computer processing and vice versa. Any background load on the VAX 11/750 system was found to have a negligible effect on the DBM benchmark results. This is a major architectural difference between the host computer database systems and a database machine.

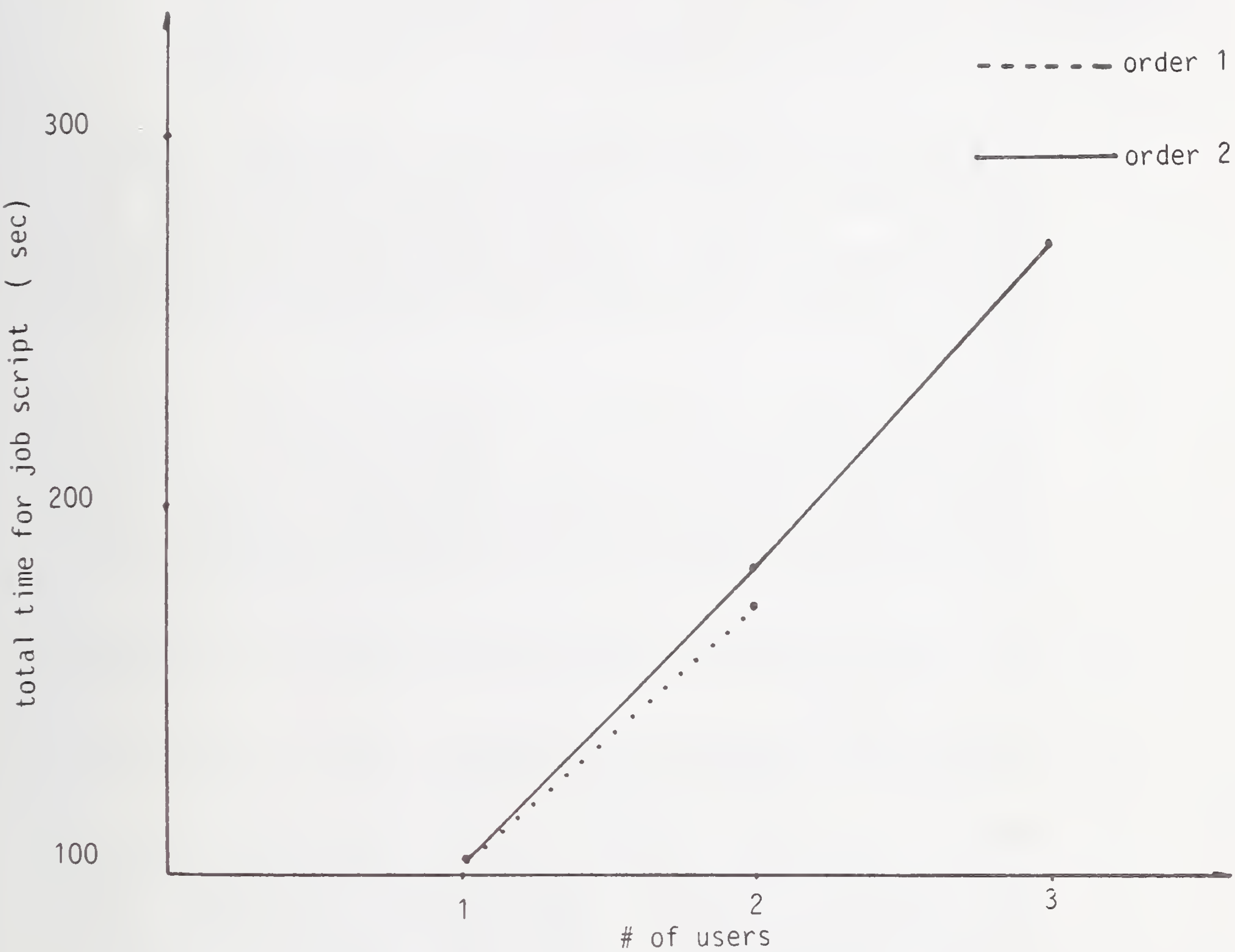


Figure 8.8: Multiple User Response Times (6 MB Database).

8.3 Summary

The following points summarize the major results of the benchmark study of the database machine architecture. These points can be used as guidelines for evaluating performance of database machines. The first 4 points show results similar in nature to the microcomputer and minicomputer database benchmarks. A comparison of performance over the three tested architectures is presented in Section 9.

1. In general, query complexity was inversely proportional to the number of records retrieved. Time-to-first-record was greater when fewer records were retrieved, since the system took longer to find the first record. Time-to-last-record was greater when a larger number of records were retrieved, due to the time required to access and transmit additional records.

2. Indexing reduces response time if the index can be used effectively in the query access strategy. Several types of queries were identified that did not allow the index to be used effectively. These cases include queries with high hit ratios, queries with disjunctive conditions, and queries with range conditions.

3. The order in which queries are run had no influence on performance.

4. Sorting and aggregation functions added significant overhead to query response time.

5. Multiple relation queries (join queries) required considerably more time to complete than single relation queries. Queries involving four relations would not complete. Database size had a significant effect on the performance of multiple relation queries. Results on the 8 MB database showed markedly decreased performance over the results on the 6 MB database.

6. The arrangement of conditions in a query had no effect on performance.

7. Writing a query with explicit join conditions or implicit join conditions had no effect on performance.

8. Update performance was significantly improved by having clustered indexes on the primary keys in a relation.

9. The presence of multiple users had a linear effect on the performance of the job scripts in the database system.

10. Background loads had no effect on the performance of queries in a back-end database machine.

9. COMPARATIVE BENCHMARK ANALYSIS

The benchmark results from the three database systems support several significant comparisons of the architectures that the systems represent. The three architectures discussed in this section are a microcomputer system (MRS), a minicomputer system (ORACLE), and a database machine (IDM-500). First, it is important to note that a majority of the benchmark studies found similar performance patterns over all three architectures. In summary, the following general observations can be made for each of the benchmarked systems.

1. Query Type. Each system supports all the query types that were tested; single relation queries, multiple relation queries, updates, sort queries, and aggregation queries.

2. Order of Query Execution. This factor had no effect on response time in any system tested, except for the phenomenon mentioned in Section 8.2.4.

3. Indexing. The use of indexing was found to be valuable for queries that used the indexes effectively. Cases were recognized, however, where indexing did not improve system performance and, in some cases, actually decreased performance. These observations were made across all systems.

4. Sorting. The addition of sorting to a query was costly for all systems.

5. Aggregations. Aggregation functions also decreased the performance of queries in all systems.

6. Special Case Studies. While several interesting observations can be made from the special case studies of rearrangement of query conditions, implicit vs. explicit join conditions, and simulated optimization methods, performance was a result more of the particular system implementation than of the system architecture. Therefore, it is felt that no major architectural differences can be cited.

Based upon the benchmark results, seven areas were found in which the database system architecture had a significant effect upon the capabilities and performance of the database system. These areas are:

1. Query Complexity
2. Number of Records Retrieved
3. Database Size
4. Number of Users
5. Background Load
6. Database System Loading and Set-Up
7. System Reliability.

The first five factors had a pronounced effect upon the performance data that were able to be gathered in the benchmark tests. The final two areas affected the execution of the benchmark. The architectural comparisons for these factors are based upon the data and experiences obtained from running the benchmarks.

9.1 Query Complexity

The principal architectural difference was that the microcomputer database system was not able to handle join queries with more than two relations in a non-procedural manner. A user is required to form such a query with nested blocks. The benchmark found that the placement of different size relations in the inner and outer blocks had a significant impact on the performance of the query. It was also found that all of the tested systems could not run the tested four relation query set. In both the single relation and multiple relation tests, query complexity directly affected the number of records retrieved. More complex queries retrieved fewer records because of the more selective conditions.

9.2 Number of Records Retrieved

Two sets of comparisons were made under this topic. First, the performance of the microcomputer architecture is compared with the minicomputer architecture. This study concentrated on the effect of differing architectural capacities of host computer database systems. In the second study the performance effects between the front-end minicomputer database system versus the back-end database machine architecture are studied.

9.2.1 Microcomputer vs. Minicomputer Architecture.

The microcomputer and minicomputer architectures were compared on the 3.5 MB database by varying the number of records retrieved by the test queries. Figure 9.1 and Figure 9.2 show the performance differences between the microcomputer database system and the minicomputer database system on time-to-first record and time-to-last record, respectively. Note that the minicomputer performance data were used as the base of the graphs.

The minicomputer database system had a significantly better time-to-first response time for all queries tested (Figure 9.1). For time-to-last, the minicomputer database system also showed faster response time for most queries (Figure 9.2). It was interesting to see that the microcomputer database system actually ran several queries faster than the minicomputer database system. Before drawing any conclusions on this comparison the following points must be mentioned:

1. The performance of a system reflected the implementation techniques used. A given system's performance changed if the implementation was improved.

2. The microcomputer database system was run on a much slower and lower priced computer than that of the minicomputer database system. One can expect a very significant performance gain if the microcomputer database system was benchmarked on a minicomputer.

3. The minicomputer database system was a more complex system with several capabilities the microcomputer did not offer; such as concurrency control for multiple users, the ability to process more complex queries, and recovery features.

Thus, it seemed that the relative simplicity of the microcomputer database system allowed it to be faster for certain queries. In most cases, however, the minicomputer system provided significantly better response times. The extensive capabilities also set the minicomputer system apart from the microcomputer system.

9.2.2 Minicomputer vs. Database Machine Architecture.

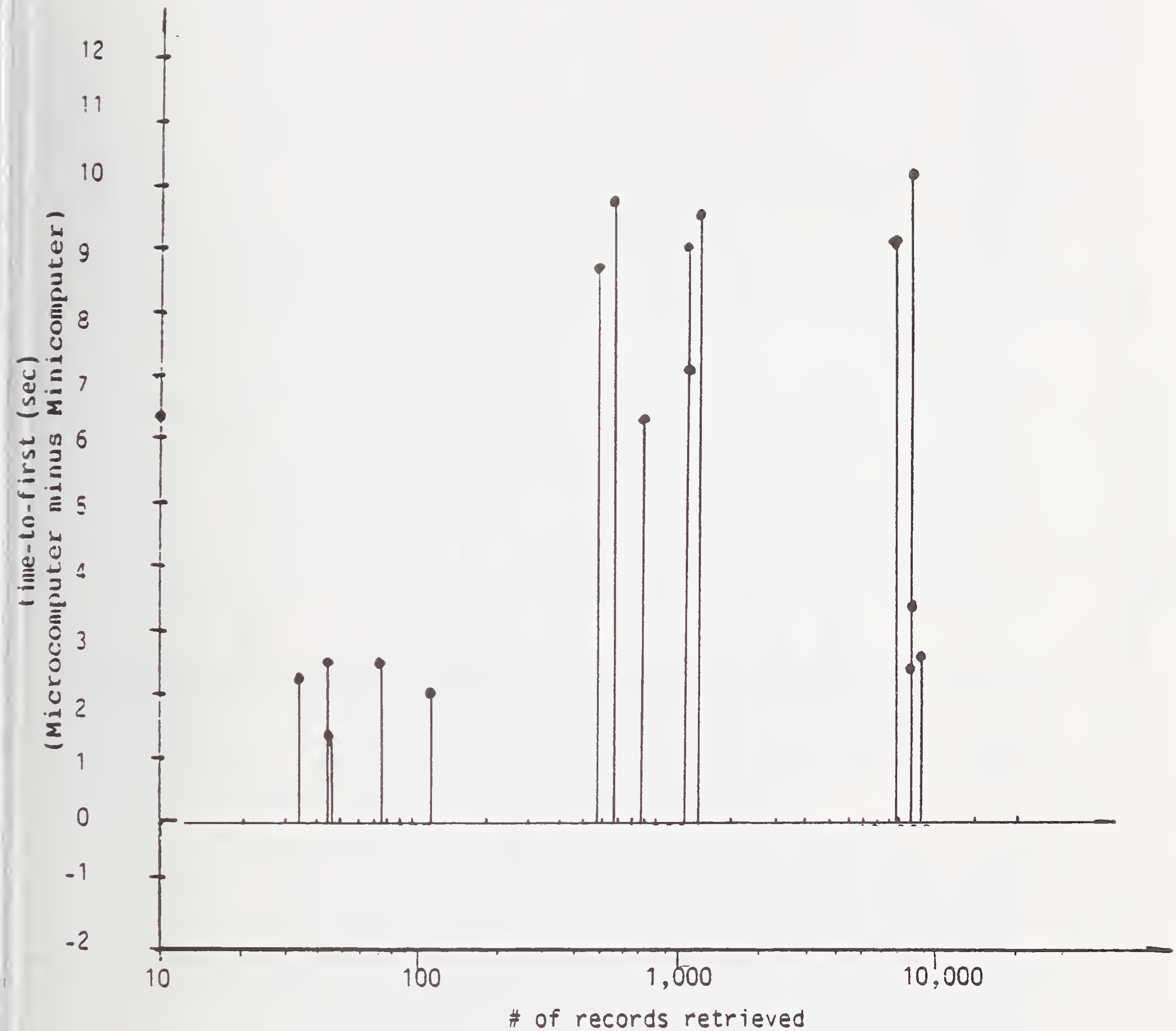


Figure 9.1: Microcomputer vs. Minicomputer Architectures
Time-to-First Difference (3.5 MB Database, Level 2 Indexes)

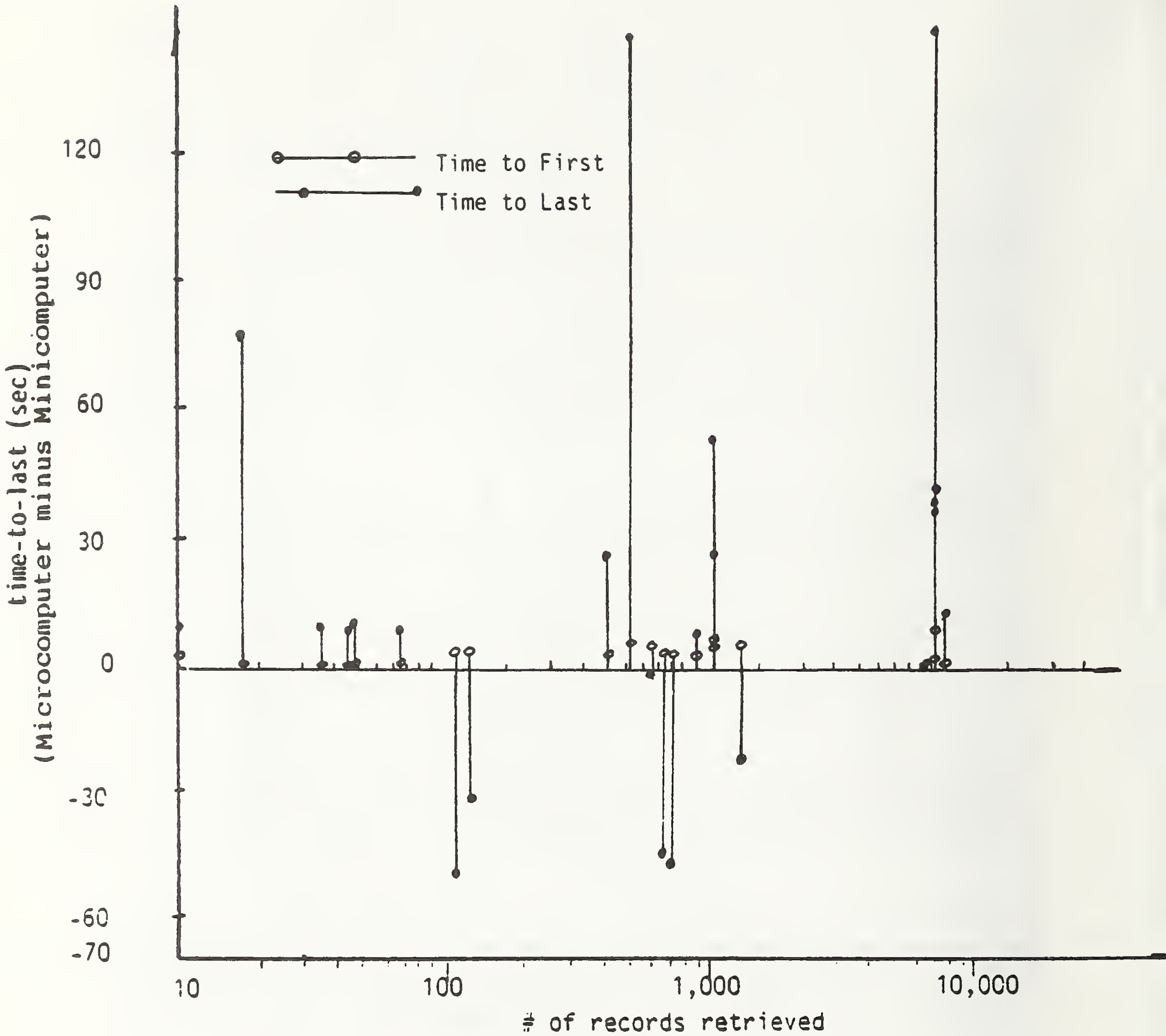


Figure 9.2: Microcomputer vs. Minicomputer Architectures
Time-to-Last Difference (3.5 MB Database, Level 2 Indexes)

The comparisons between the minicomputer architecture and the database machine architectures provided some very interesting observations. The database machine performance data was used as the base of the graphs in this section. Figures 9.3 and 9.4 show the time-to-first record and time-to-last record data for the single relation queries on the 3.5 MB database using level 2 indexes. Figure 9.3 shows that in every query the minicomputer architecture had better time-to-first performance than the database machine architecture. This clearly illustrated the effect of the 9600 baud data channel that connected the front-end host computer with the database machine. The delay of sending the query to the database machine and receiving the first record from the machine imposed a significant delay for the time-to-first value across all queries independent of the number of records eventually retrieved.

In Figure 9.4, the time-to-last data demonstrated the advantages of the database machine architecture. In order to compare the scales of Figures 9.3 and 9.4, the time-to-first data has been included on both graphs. It is clear that the time-to-first performance disadvantage of the database machine architecture was not significant when compared with its advantage in the time-to-last performance. Of course, the relative importance of the time-to-first and the time-to-last performance was dependent upon the particular application.

Note that the amount of the performance difference changed as the number of records retrieved varied. For queries that retrieved less than 100 records the performance difference was negligible. Note that both systems had level 2 indexes. Thus, fast retrieval of the small set of result records was expected on both architectures.

The major performance difference was observed in the medium range of records retrieved, between 100 and 2000 records. Here the advantages of a specialized hardware and software database system in a back-end machine were apparent. The database machine had significantly better time-to-last performance for this range of queries. For the range of records retrieved between 5000 and 10,500 records the performance differences were relatively insignificant. The reason was that the total time-to-last for retrieving this number of records was quite large. In relationship to the great response time, the performance differences between the two architectures were not significant.

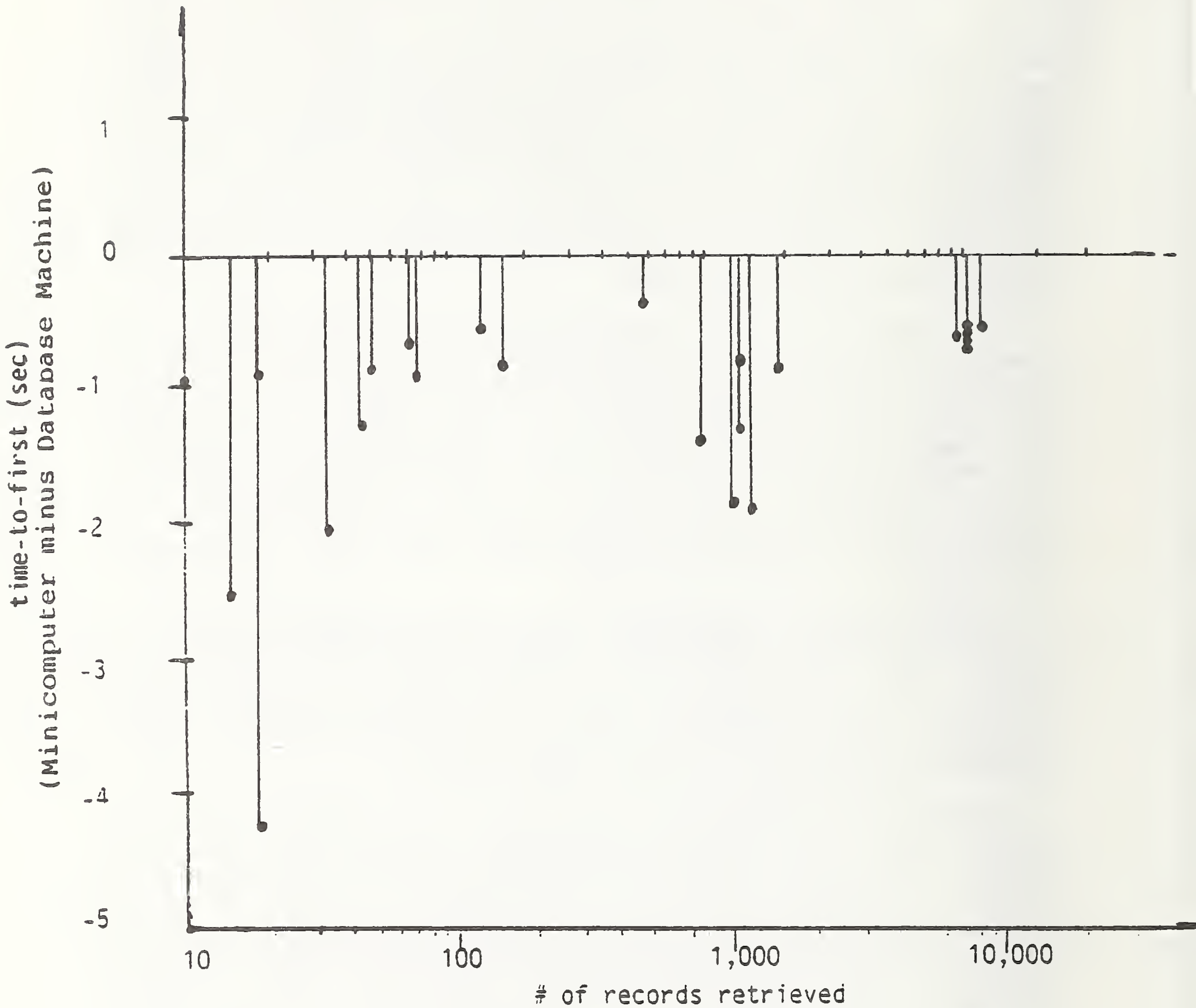


Figure 9.3: Minicomputer vs. Database Machine Architectures
Time-to-First Difference (3.5 MB Database, Level 2 Indexes)
Single Relation Queries

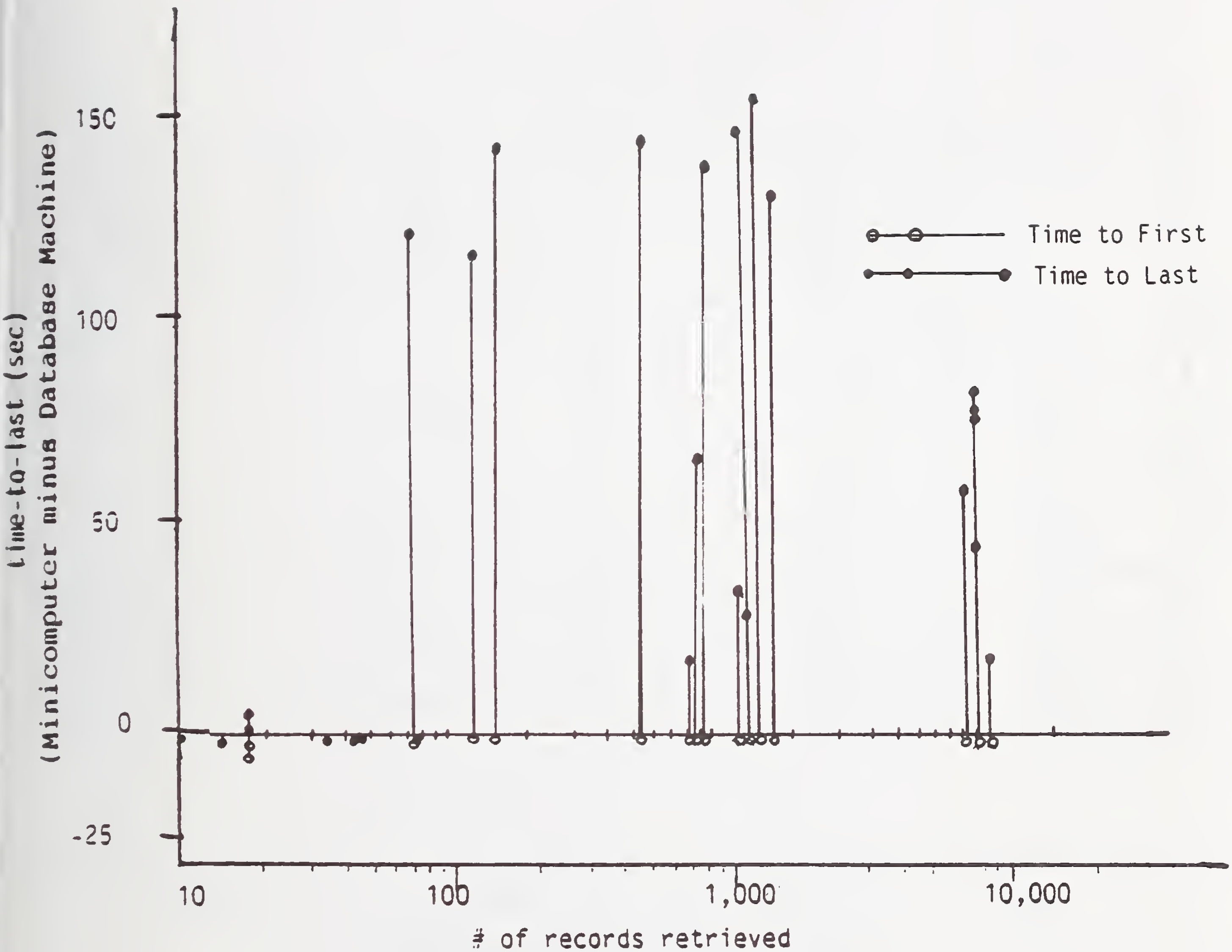


Figure 9.4: Minicomputer vs. Database Machine Architectures
Time-to-Last Difference (3.5 MB Database, Level 2 Indexes)
Single Relation Queries

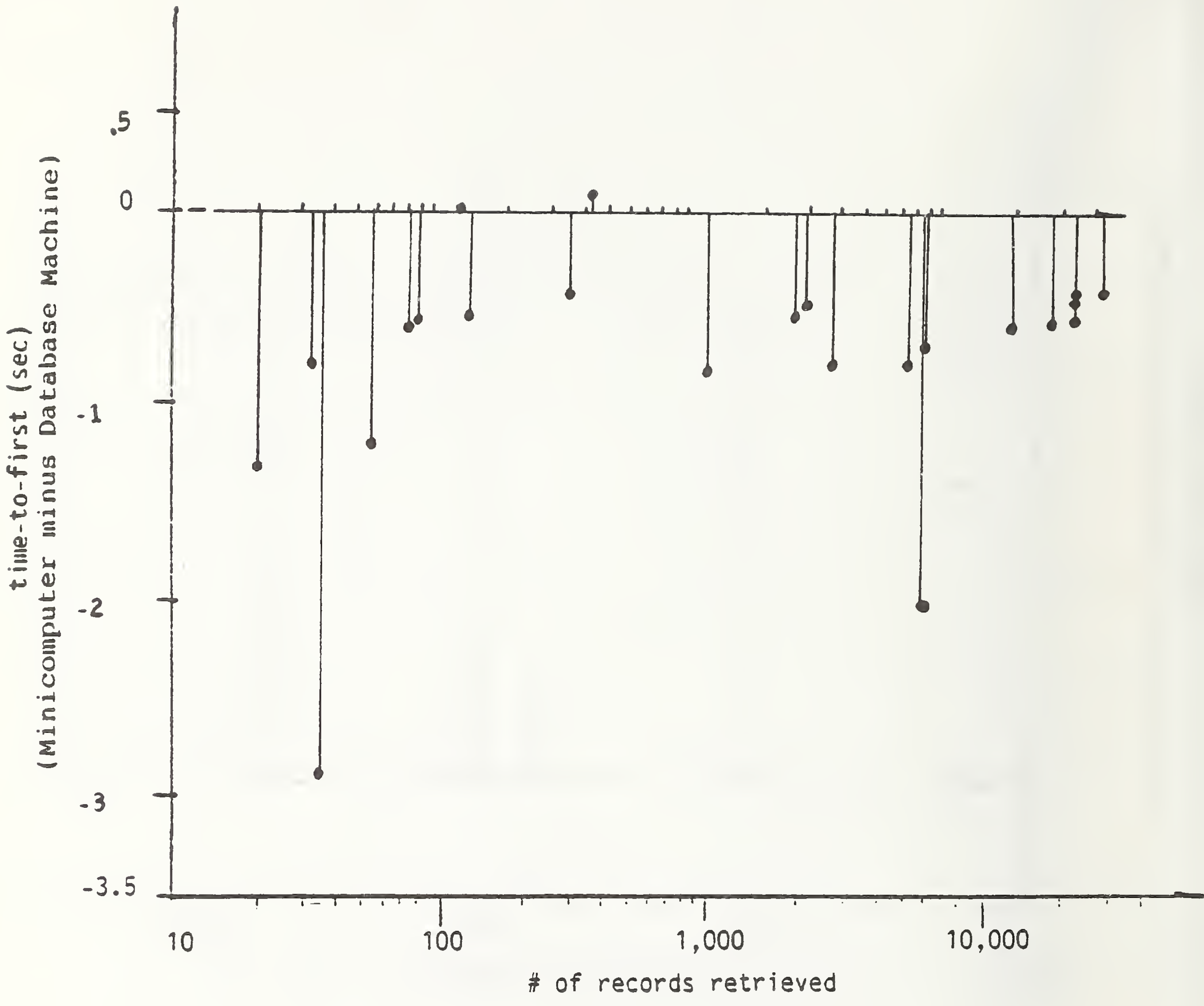


Figure 9.5: Minicomputer vs. Database Machine Architectures
Time-to-First Difference (10 MB Database, Level 2 Indexes)
Single Relation Queries

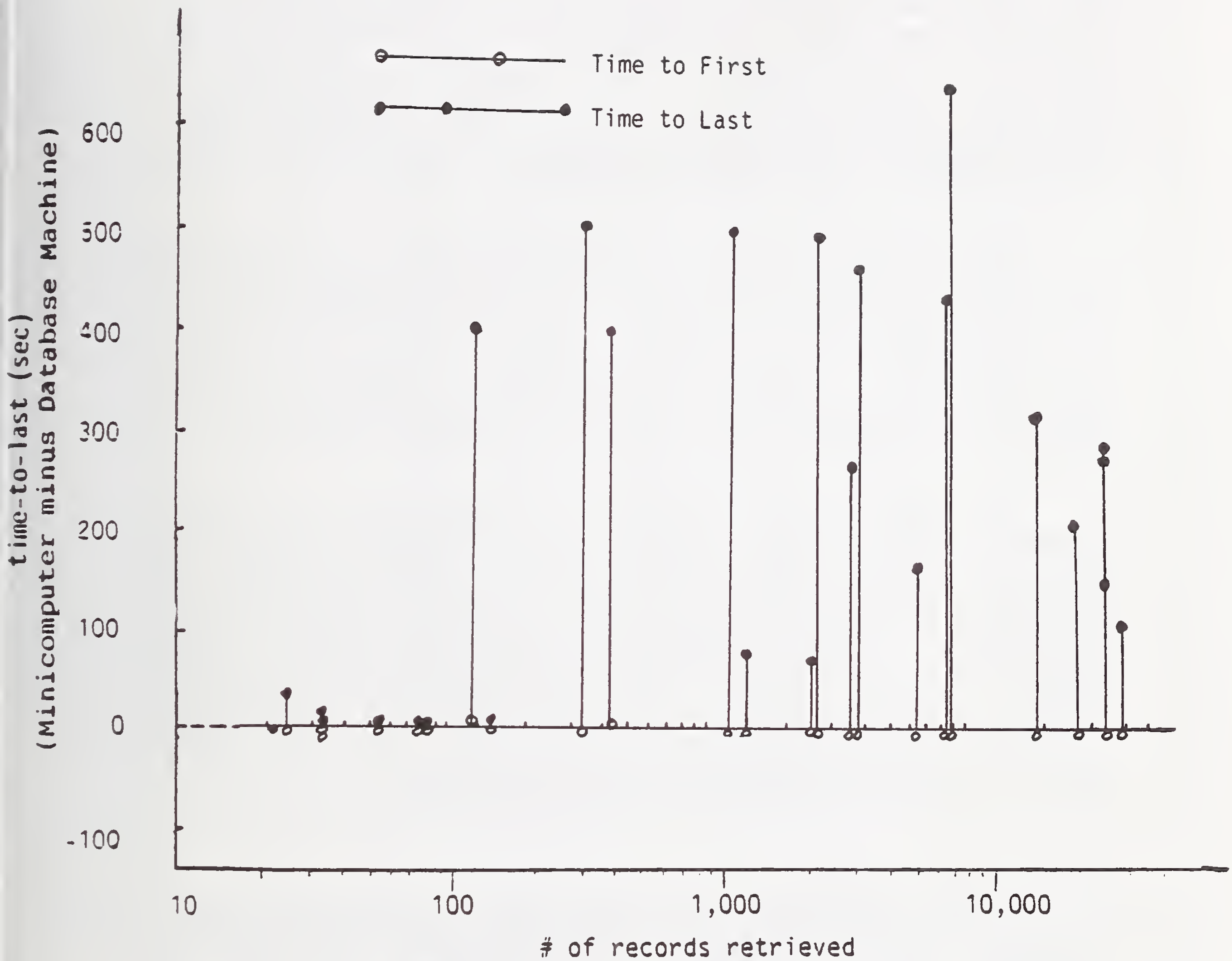


Figure 9.6: Minicomputer vs. Database Machine Architectures
Time-to-Last Difference (10 MB Database, Level 2 Indexes)
Single Relation Queries

The data in Figures 9.5 and 9.6 demonstrate that the above observations also held for the performance data found on the 10 MB database with level 2 indexes. The minicomputer database system had better time-to-first response time over the complete range of records retrieved (Figure 9.5). The database machine architecture had better time-to-last response time than the minicomputer architecture (Figure 9.6). As on the 3.5 MB database, the performance difference varied based upon whether a small number of records was retrieved (1-100 records), a medium number of records was retrieved (200-10,000 records), or a large number of records was retrieved (10,000-33,000 records). The largest relation in the 10 MB database contained 33,000 records.

Next the performance differences between the minicomputer architecture and the database machine architecture for the multiple relation queries on the 6 MB database with level 2 indexes were studied. The time-to-first data is graphed in Figure 9.7 and the time-to-last data is graphed in Figure 9.8. Again, the minicomputer system had a better time-to-first performance. Due to the complexity of join processing, however, this better performance was not as consistent as in the single relation cases.

The time-to-last data did not demonstrate the differences in performance (based upon the number of records retrieved) that were seen in the single relation graphs. The database machine architecture had a consistently better performance across the entire range from 1 to 20,000 records retrieved. It was concluded that multiple relation queries required the use of architecturally dependent access strategies and the result was independent of the number of records eventually retrieved. Therefore, the speed advantages of the database machine architecture were observed for all of the tested multiple relation queries.

9.3 Database Size

In the benchmarks, the size of the database was varied from 3.5 MB to 56 MB. The database machine was tested on database sizes of 3.5 MB, 6 MB, 8 MB, 10 MB, and 56 MB. The minicomputer database system was tested on database sizes of 3.5 MB, 6 MB. and 10 MB. The microcomputer database system was only able to run the 3.5 MB database. The microcomputer architecture was limited by memory capacity and storage capacity. The limited memory capacity for handling buffers and index workspace, in particular, caused many queries to be aborted. The microcomputer architecture could not support databases of the size that could be handled easily by the minicomputer and database machine architectures.

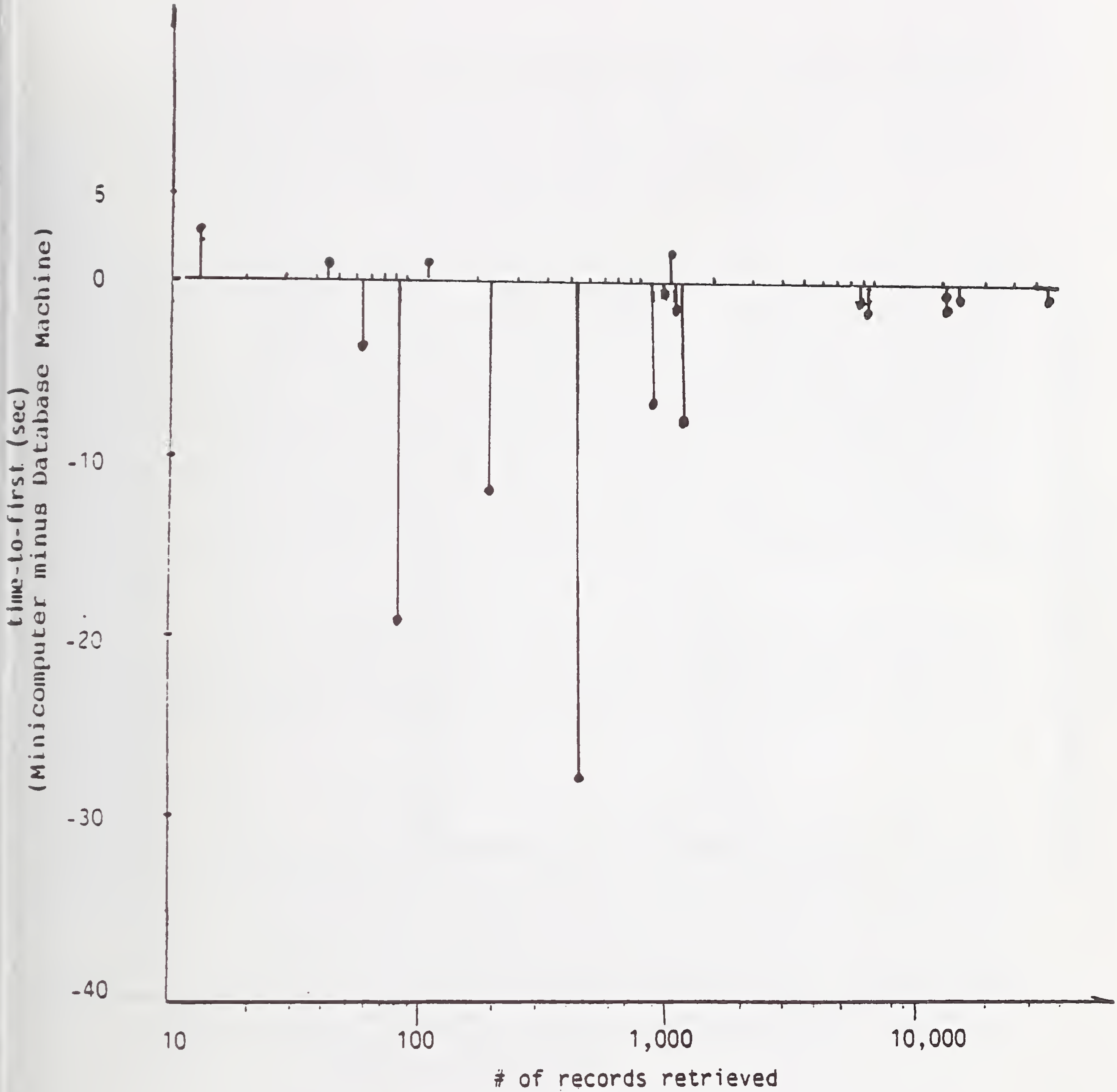


Figure 9.7: Minicomputer vs. Database Machine Architectures
Time-to-First Difference (6 MB Database, Level 2 Indexes)
Multiple Relation Queries

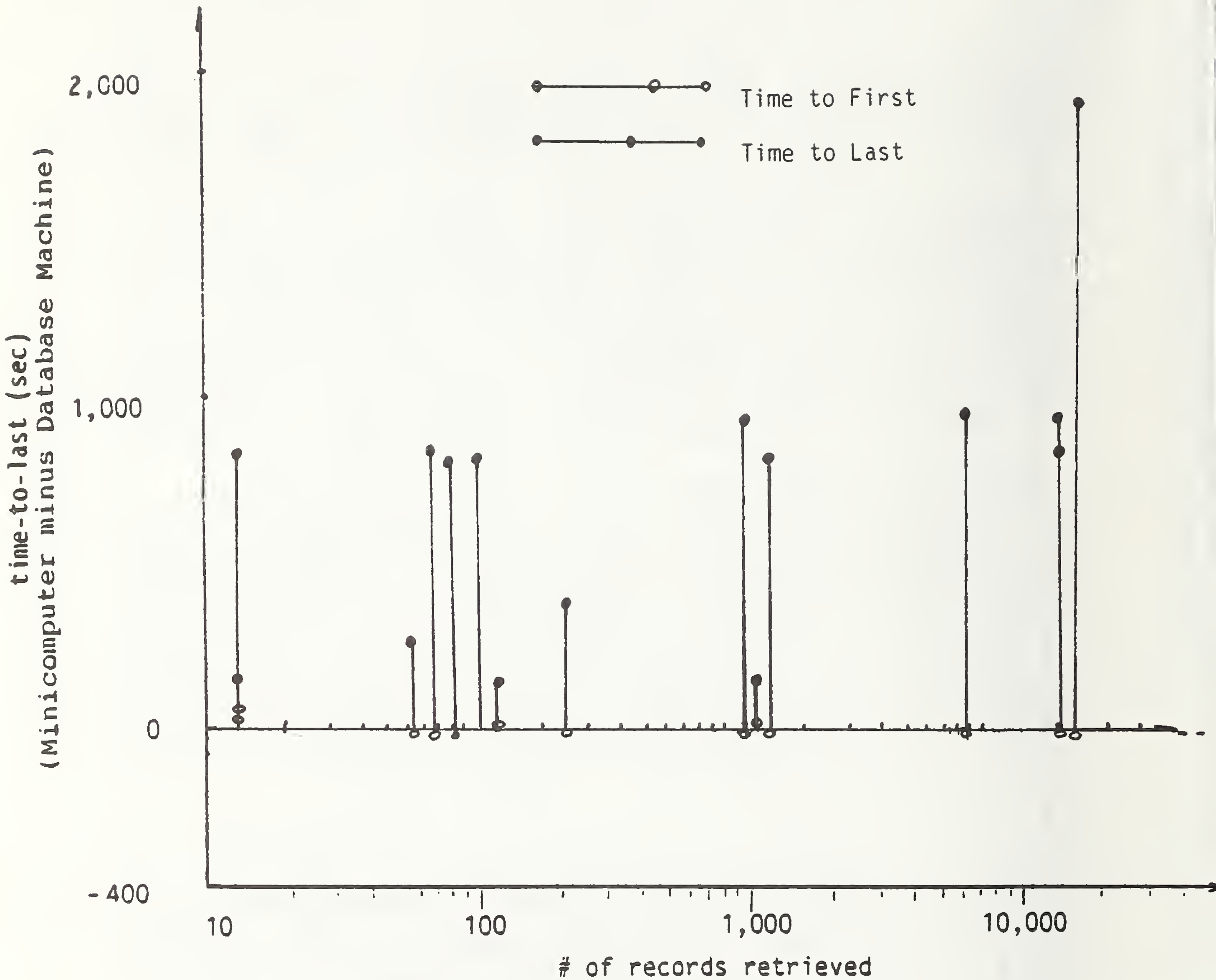


Figure 9.8: Minicomputer vs. Database Machine Architectures
Time-to-Last Difference (6 MB Database, Level 2 Indexes)
Multiple Relation Queries

A significant difference was not found in the abilities of the minicomputer database system and the database machine to support database sizes in the range that was tested. While tests for the 56 MB database were run on the database machine, the results were so inconclusive (e.g., no join queries ran to completion within 30 minutes) that a similar set of tests on the minicomputer system was not run.

In Figure 9.9 the average response time in seconds for single relation queries for each system over the tested database sizes is shown. The average was taken over all queries tested using equal weights. In Figure 9.10 the average response time in seconds for multiple relation queries for each system is shown. For the graph in Figure 9.10 certain response time values for queries that exceeded 30 minutes were extrapolated from measured performance data. The extrapolation was performed by recording the percentage increase in average response time on query set 6 between the 6 MB and 8 MB databases and the 6 MB and 10 MB databases. Then this percentage increase was applied to the average response time over all multiple relation query sets on the 6 MB database to find the estimated average response times in Figure 9.10 for the 8 MB and 10 MB databases.

It was concluded that both the minicomputer and database machine architectures tested can support databases from 3.5 MB to 10 MB with varying degrees of efficiency. The database machine seemed to have better performance for the cases tested.

9.4 Number of Users

Multiple user tests were run on the minicomputer and database machine database systems. The microcomputer system was tested as a dedicated system; architecturally unable to support more than one user. Figure 9.11 shows the response time in seconds for a specially designed job script on the two systems as the number of users varies from 1 to 3. Both architectures can support multiple users. The performance of both systems degraded linearly with the number of users in the system. Thus no architectural differences were observed.

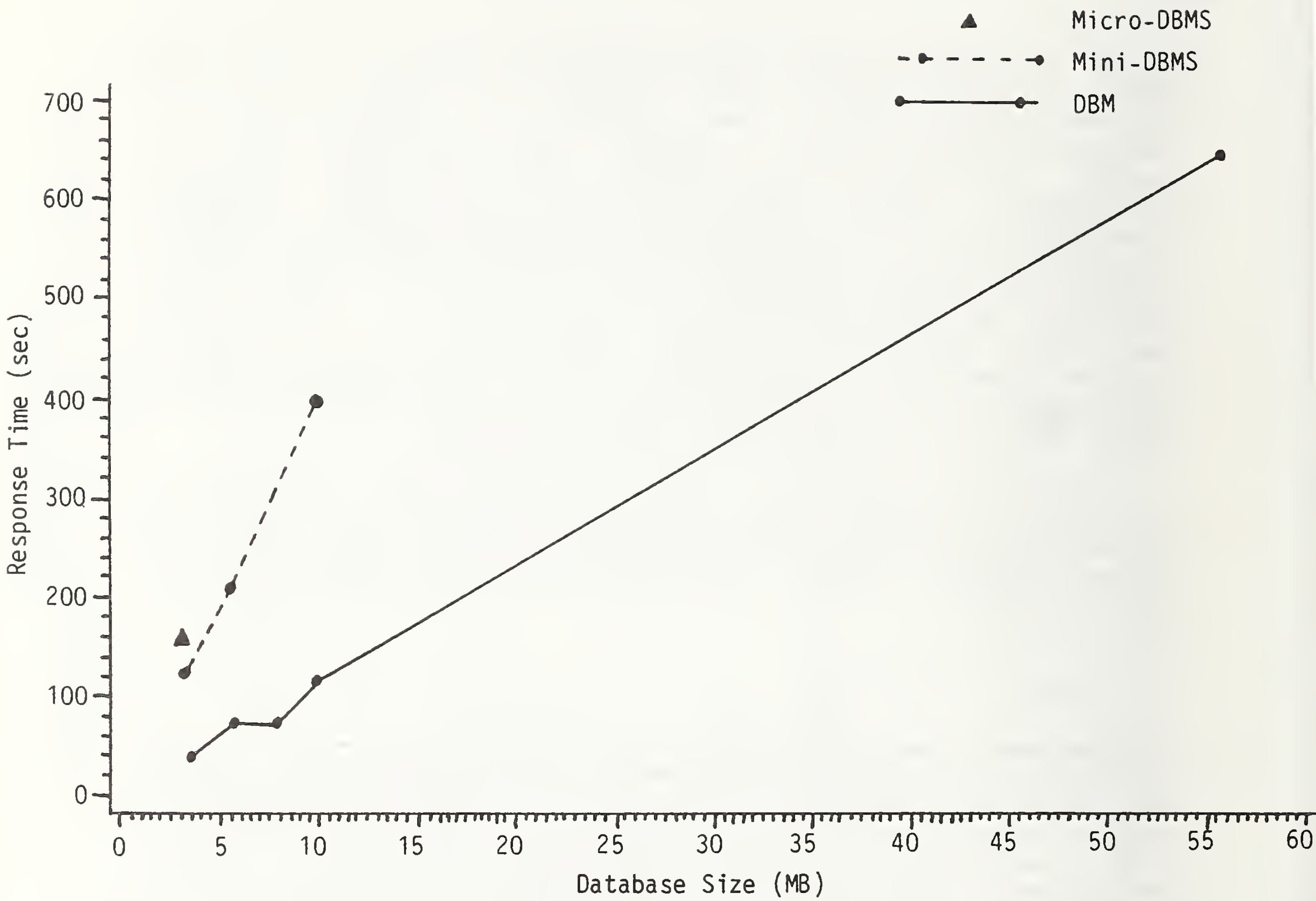


Figure 9.9: Database Size vs. Average Response Time
Single Relation Queries

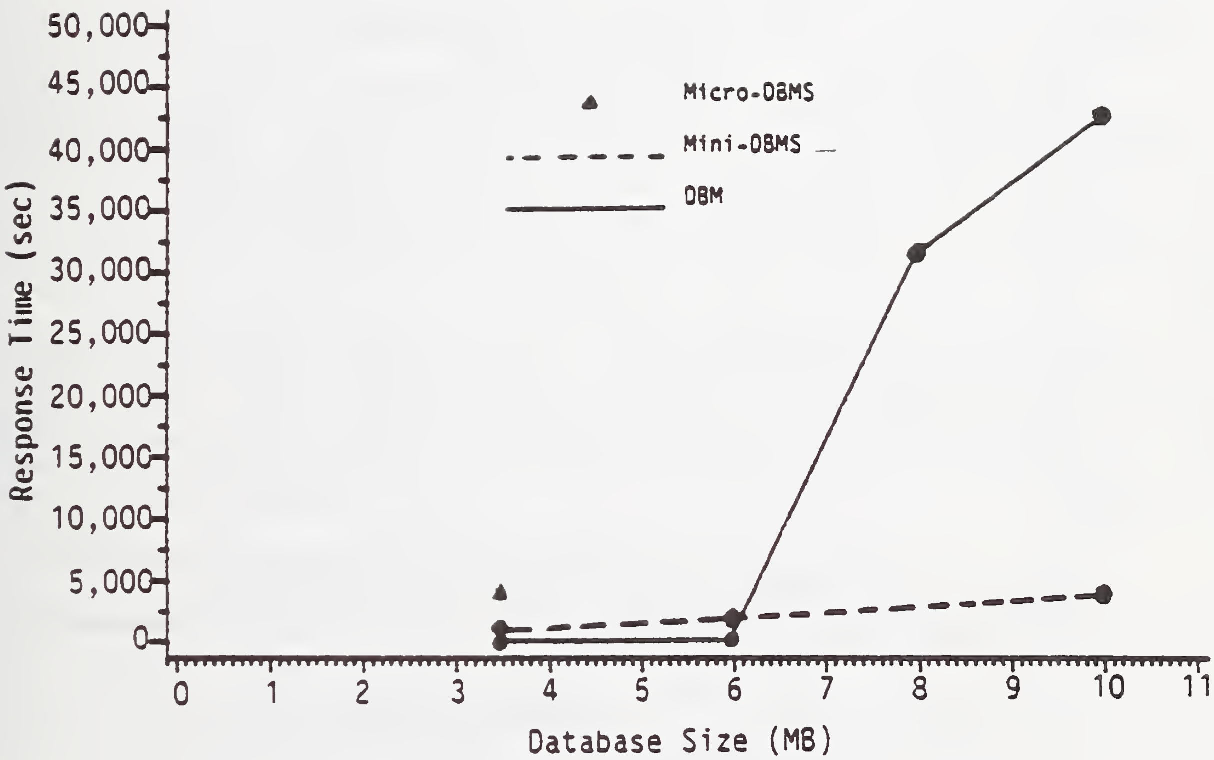


Figure 9.10: Database Size vs. Average Response Time
Multiple Relation Queries

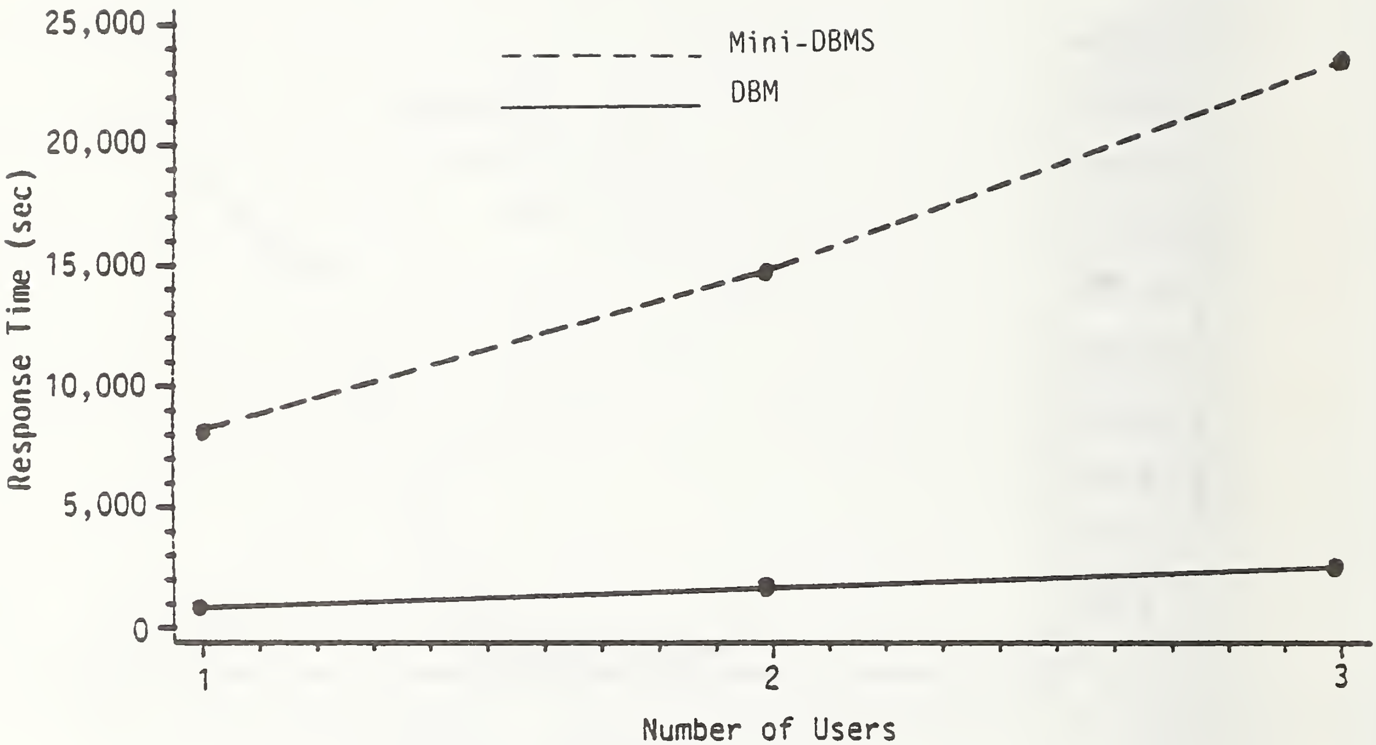


Figure 9.11: Multi-User Architecture Comparison

9.5 Background Load

One of the major advantages of the database machine architecture was that it relieved the front-end computer system of database processing. The effect of the load on the front-end VAX 11/750 was found to have a negligible performance effect on the database machine processing. The background load tests on the minicomputer database system resulted in a significant increase in the job script response time as more background jobs were added on the host VAX 11/750. Figure 9.12 illustrates the performance effects of adding background jobs for the two tested architectures.

9.6 Database Loading and System Set-Up

The database system loading and set-up procedures varied greatly among the three systems. The microcomputer database system required the least expertise and set-up time. A user could master the techniques for loading and system initialization in a short time. The other two systems required a significant effort for loading and set-up.

For the minicomputer database system several minor set-up delays were experienced. These problems were quickly solved by means of phone calls to the vendor's systems staff. The load procedure of reading the database from a tape to disk went smoothly. While testing queries, buffer space kept running out. This was remedied by increasing the size of a work space file.

The database machine system architecture was the most difficult to load and set-up. Considerable time was spent in order to configure the database machine with a compatible disk drive. The vendor's system personnel had to come on site to help solve these problems. The data loading procedure was extremely time consuming. Loading the 56 MB database required over 26 hours. The bottleneck was the 9600 baud communications line between the VAX 11/750 and the database machine. Faster communication links between the back-end and the host computer are needed for more reasonable data transfer time.

9.7 System Reliability

The basic architectural issue for reliability was the presence of two computer systems in a database machine architecture. Both systems had to function correctly for the database machine to be operative. This effectively decreased the reliability of a database machine architecture.

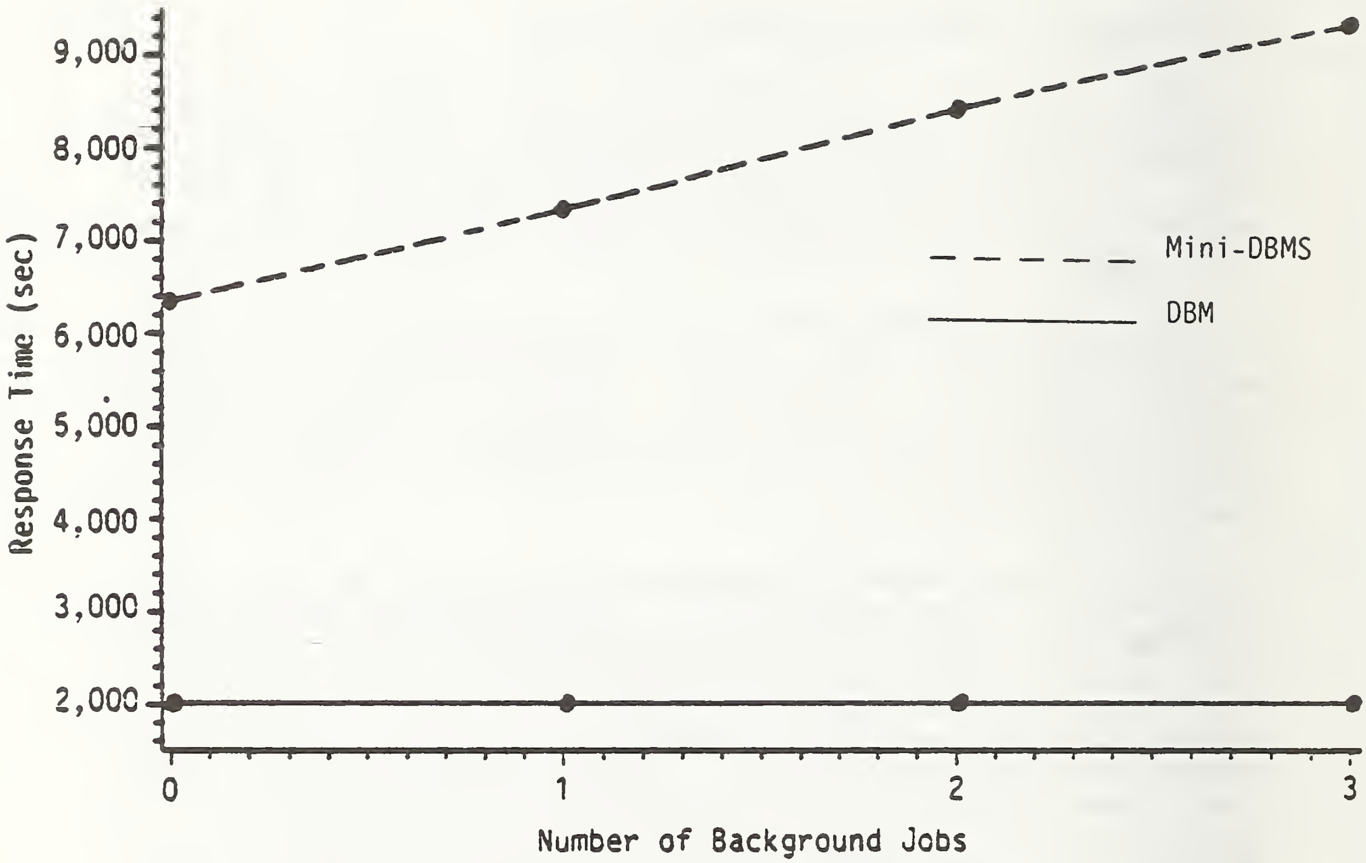


Figure 9.12: Background Load Architecture Comparison

For example, if the probability of failure of the VAX 11/750 is p(i) and the probability of failure of the back-end system is p(j), then the probability of failure for the complete front-end/back-end system would be

$$1 - (1 - p(i)) * (1 - p(j)) = p(i) + p(j) - p(i)p(j).$$

Such an increase in failure probability would be significant in an environment that requires high reliability of its database system.

During the benchmark experiments, an instance was experienced where a storm caused the database machine to crash while the VAX 11/750 was unaffected. As a result the database was unavailable until the database machine was rebooted the next day.

9.8 Summary

The comparisons made in this section clearly show that each of the three database system architectures has definite advantages and disadvantages in terms of performance and usability. It was also found that the different architectures exhibit similar characteristics in some performance parameters. While understanding the risk of over simplification, the major observations are summarized briefly in the following points:

1. Microcomputer Database Systems.

   Microcomputer systems have limited capacities for data storage and workspace buffers. Joining capabilities may be limited to at most two relations in a query. The principal advantages of microcomputer database systems are their simplicity and ease of use.

2. Minicomputer Database Systems.

   Minicomputer database systems performed effectively on databases of size from 3.5 MB to 10 MB. These systems offer extended capabilities beyond the microcomputer database systems. These capabilities include concurrency control for multiple users, the ability to process more complex queries, and recovery features. At least for the system benchmarked, it seemed that these additional capabilities were obtained at the expense of lower system performance in some areas. It was also found that running a database system placed a significant load on the host computer. The database jobs and background jobs were shown to compete for computer resources.

3. Database Machine Systems.

   Database machines provide a specialized hardware and software component that performs most of the database processing on the back-end of a host computer system. Thus, database processing has a negligible resource requirement on the host computer. Similarly to the minicomputer database system, the database machine performed effectively in the tested range from 3.5 MB to 10 MB databases. In most cases it provided a faster time-to-last response time and a slightly slower time-to-first response time than the minicomputer system. The database machine system was found to have a more complex set-up and load procedure and to be slightly less reliable because of the presence of two separate, dependent systems.

This report has concentrated on the evaluation of the three architectures based upon performance factors. A comprehensive evaluation procedure for selecting and evaluating a database system must include other important factors, such as cost, user friendly interfaces, documentation, add-on features (e.g., report writers, word processing, graphics packages, etc.), and other qualitative features essential to a particular operating environment. The conclusions and guidelines drawn in this report must be applied only after a thorough requirements analysis of a particular database environment and a features analysis of candidate database systems. Finally, as was emphasized in the Introduction, these results should not be heavily relied on out of context. This is a snapshot of a dynamically changing environment, particularly in the microcomputer hardware and software areas.

REFERENCES

[BENI 84] Benigni, D. (Editor), Yao, S., and Hevner, A. R., A Guide to Performance Evaluation of Database Systems, NBS Special Publication 500-118, 1984.

[CARD 73] Cardenas, A. "Evaluation and Selection of File Organization - A Model and System," Communications of the ACM, Vol. 16, No. 9, Sept. 1973.

[CODD 82] Codd, E. "Relational Database: A Practical Foundation for Productivity," Communications of the ACM , Vol. 25, No. 2, February 1982.

[DATE 81] Date, C. An Introduction to Database Systems Third Edition, Addison-Wesley Inc., 1981.

[GALL 84] Gallagher, L.J., and Draper, J.M. Guide on Data Models in the Selection and Use of Database Management Systems, NBS Special Publication 500-108, January 1984.

[IDM 82a] IDM-500 Installation and Operation Manual, 201-0500, March 1982.

[IDM 82b] IDM-500 Software Reference Manual, Version 1.4, 202-0500, 1982.

[LUO 82] Luo, D., Xia, D. and Yao, S. "Data Language Requirements for a Database Machine," Proceedings of NCC, 1982.

[MAKI 82] Makimo, T. et al. "An Evaluation of a Generalized Database Subsystem," Journal of Information Processing , Vol. 5, No. 1, March 1982.

[NBS 80] NBS Guideline for Planning and Management of Database Applications, FIPS PUB 77, September 1980.

[NBS 84] NBS Guideline for Choosing a Data Management Approach, FIPS PUB 110, December 1984.

[ORAC 83] ORACLE Database System Manuals, Version 3.1
Oracle Database Administrators Guide,
UFI Terminal Users Guide,
UFI Terminal Users Reference Manual,
SQL/UFI Reference Manual,
HLI Call Interface Manual,
Oracle Database Administrators Guide,
Database Loader Utility - ODL,

Database Backup and Recovery,
Oracle VMS Installation Guide,
Oracle Error Messages and Codes,

[STON 80] Stonebraker, M. "Retrospective on a Database System," ACM Transactions on Database Systems, Vol. 5, No. 2, 1980.

GLOSSARY

ACCESS. The operation of seeking, reading, or writing data on a storage unit.

ACCESS METHOD. A technique for moving data between a computer and its peripheral devices, e.g., serial access, random access, remote access, virtual sequential access method (VSAM), hierarchical indexed sequential access method (HISAM).

ACCESS TIME. The time that elapses between an instruction being given to access some data and that data becoming available for use.

ADDRESS. An identification (number, name, label) for a location in which data is stored.

ATTRIBUTE. A field containing information about an entity.

AVAILABILITY. A measure of the compatibility of a system to be used for performing its intended function, as a result of the system´s being in an operating state.

BLOCKING. The combining of two or more records so that they are jointly read or written by one machine instruction.

BUFFER. An area of storage which holds data temporarily while it is being received, transmitted, read or written. It is often used to compensate for differences in speed or timing of devices. Buffers are used in terminals, peripheral devices, storage units and in the CPU.

CHANNEL. A subsystem for input to and output from the computer. Data from storage units, for example, flows into the computer via a channel.

DATABASE. A collection of interrelated data stored together with controlled redundancy to serve one or more applications; the data are stored so that they are independent of programs which use the data; a common and controlled approach is used in adding new data and in modifying and retrieving existing data within a database. A system is said to contain a collection of databases if they are disjoint in structure.

DATABASE MANAGEMENT SYSTEM. The collection of software required for using a database.

DATA DICTIONARY. A catalogue of all data types giving their names and structures.

DATA ITEM. The smallest unit of data that has meaning in describing information; the smallest unit of named data. Synonymous with DATA ELEMENT or FIELD.

DATA MANAGEMENT. A general term that collectively describes those functions of the system that provide creation of and access to stored data, enforce data storage conventions, and regulate the use of input/output devices.

DIRECT ACCESS. Retrieval or storage of data by a reference to its location on a volume, rather than relative to the previously retrieved or stored data.

DIRECTORY. A table giving the relationships between items of data. Sometimes a table (index) giving the addresses of data.

DOMAIN. The collection of data items (fields) of the same type, in a relation (flat file).

ENTITY. Something about which data is recorded.

FILE. A set of similarly constructed records.

FUNCTIONAL DEPENDENCE. Attribute $\underline{B}$ of a relation $\underline{R}$ is functionally dependent on attribute $\underline{A}$ or $\underline{R}$ if, at every instant in time, each value of $\underline{A}$ has no more than one value of $\underline{B}$ associated with it in relation $\underline{R}$.

HIT RATE. A measure of the number of records in a file which are expected to be accessed in a given run. Usually expressed as a percentage:

$$\frac{\text{Number of input transaction x 100\%}}{\text{Number of records in the file}}$$

INDEX. A table used to determine the location of a record.

INDEX, CLUSTERED. Records in a file are physically organized based upon the values of the indexed attribute.

INDEX, UNCLUSTERED. The indexed attribute does not effect the physical storage of records in a file.

KEY. A data item used to identify or locate a record (or other data grouping).

KEY, PRIMARY. A key which uniquely identifies a record (or other data grouping).

KEY, SECONDARY. A key which does not uniquely identify a record, i.e., more than one record can have the same key value. A key which contains the value of an attribute (data item) other than the unique identifier.

MODEL. The logical structure of the data. Schema.

MULTILIST ORGANIZATION. A chained file organization in which the chains are divided into fragments in each fragment indexed, to permit faster searching.

MULTIPLE-KEY RETRIEVAL. Retrieval which requires searches of data based on the values of several key fields (some or all of which are secondary keys).

NORMAL FORM, FIRST. Data in flat file form.

NORMAL FORM, SECOND. A relation R is in second normal form if it is in first normal form and every nonprime attribute of R is fully functionally dependent (q.v.) on each candidate key of R (E. F. Codd's definition).

NORMAL FORM, THIRD. A relation R is in third normal form if it is in second normal form and every nonprime attribute of R is nontransitively dependent on each candidate key of R (E. F. Codd's definition).

NORMALIZATION. The decomposition of more complex data structures into flat files (relations). This forms the basis of relational databases.

OPERATING SYSTEM. Software which enables a computer to supervise its own operations, automatically calling in programs, routines, language, and data, as needed for continuous throughput of different types of jobs.

PAGING. In virtual storage systems, the technique of making memory appear larger than it is by transferring blocks (pages) of data or programs into that memory from external storage when they are needed.

POINTER. The address of a record (or other data groupings) contained in another record so that a program may access the former record when it has retrieved the latter record. The address can be absolute, relative, or symbolic, and hence the pointer is referred to as absolute , relative, or symbolic.

RANDOM ACCESS. To obtain data directly from any storage location regardless of its position with respect to the previously referenced information. Also called DIRECT ACCESS.

RANDOM ACCESS STORAGE. A storage technique in which the time required to obtain information is independent of the location of the information most recently obtained. This strict definition must be qualified by the observation that we usually mean relatively random. Thus, magnetic drums are relatively non-random access when compared to magnetic cores for main memory, but relatively random access when compared to magnetic tapes for file storage.

RELATION. A flat file. A two-dimensional array of data elements. A file in normalized form.

RELATIONAL ALGEBRA. A language providing a set of operators for manipulating relations.

RELATIONAL CALCULUS. A language in which the user states the results he requires from manipulating a relational data base.

RELATIONAL DATABASE. A database made up of relations (as defined above). Its database management system has the capability to recombine the data elements to form different relations thus giving great flexibility in the usage of data.

SCHEMA. A map of the overall logical structure of a data base.

SECONDARY INDEX. An index composed of secondary keys rather than primary keys.

SECONDARY STORAGE. Storage facilities forming not an integral part of the computer but directly linked to and controlled by the computer, e.g., disks, magnetic tapes, etc.

SEQUENTIAL PROCESSING. Accessing records in ascending sequence by key; the next record accessed will have the next higher key, irrespective of its physical position in the file.

SORT. Arrange a file in sequence by a specified key.

TABLE. A collection of data suitable for quick reference, each item being uniquely identified either by a label or by its relative position.

THIRD NORMAL FORM. A record, segment, or tuple, which is normalized (i.e., contains no repeating groups) and in which every nonprime data item is nontransitively dependent and fully dependent on each candidate key.

In other words: the _entire_ primary key or candidate key is needed to identify each other data item in the record and no data item is identified by a data item which is not in the primary key or candidate key.

TRANSFER RATE. A measure of the speed with which data is moved between a direct-access device and the central processor. (Usually expressed as thousands of characters per second or thousands of bytes per second.)

WORKING STORAGE. A portion of storage, usually computer main memory, reserved for the temporary results of operations.

APPENDIX A - PERSONNEL FILE FORMATS

RECORD LAYOUT FOR FILE: CPDFKOM1 RECORD SIZE: 180 Ch
WORDS: 30

FILE DESCRIPTION: CPDF Transaction History File

SORT SEQUENCE - Agency (2 POS). SSN, Eff Date, PAC

CHARACTER POSITION	DATA
1	Record Type
2-3	Agency
4-5	Subelement
6-8	PAC(Personnel Action Code)
9-14	Effective Date (YYMMDD)
15-18	SON (Submitting Office Number)
19-22	Date of Birth (YYMM)
23-26	Service Computation Date (YYMM)
27-30	SMSA (Standard Metropolitan Statistical Area)
31	PMIP (Presidential Management Intern Program)
32	Filler
33	Veterans Preference
34-35	Handicap Code
36-37	SPID (Special Program Ident.)
38	Sex
39	Tenure
40	Minority
41	Citizenship
42	PATCO
43	Pay Rate Determinant
44-52	SSN (Social Security Number)
53-56	BUS (Bargaining Unit Status)
57	FLSA (Fair Labor Stnds. Auth.)
58	VEV (Vietnam Era Veteran)
59	Annuitant Indicator
60-62	Legal Authority 1
63-65	Legal Authority 2
66-68	Current Appointment Authority 1
69-71	Current Appointment Authority 2
	Latest Data Values
72	FEGLI
73	Retirement
74	Position Occupied
75	Work Schedule
76-79	Occupational Code
80-81	Functional Classification
82-90	Geographic Location Code
91-92	Pay Basis
93-94	Pay Plan
95-96	Grade
97-98	Step
99-103	Salary
104	Supervisory Code
105-106	Educational Level
107-108	Year Degree Attained
109-112	Academic Discipline
113-114	Type of Appointment

RECORD LAYOUT FOR FILE: CPDFKOM1 RECORD SIZE: 180 Ch
WORDS: 30

FILE DESCRIPTION: CPDF Transaction History File

SORT SEQUENCE - Agency (2 POS). SSN, Eff Date, PAC

CHARACTER POSITION	DATA
	Previous Data Values
115	FEGLI
116	Retirement
117	Position Occupied
118	Work Schedule
119-122	Occupational Code
123-124	Functional Classification
125-133	Geographic Location Code
134-135	Pay Basis
136-137	Pay Plan
138-139	Grade
140-141	Step
142-146	Salary
147	Supervisory Code
148-149	Educational Code
150-151	Year Degree Attained
152-155	Academic Discipline
156-157	Type of Appointment
158-161	Date Entered Curr. Code (YYMM)
162-165	Date Entered Curr. Occup. (YYMM)
166-169	Process Date (YYMM)
170-180	Filler

RECORD LAYOUT FOR FILE: CPDFFFX3 RECORD SIZE: 72 Ch
WORDS: 30

FILE DESCRIPTION: High Utility Extract File

SORT SEQUENCE - Agency (2 positions)

CHARACTER POSITION	DATA
1-6	Service Computation Date (YYMMDD)
7-12	Date of Birth (YYMMDD)
13	Work Schedule
14-17	SON
18-22	Salary
23-26	Occupation
27-30	City
31-33	County
34-37	SMSA
38-39	Handicap
40	PRD
41-42	Pay Basis
43	Veterans Preference
44	Tenure
45	Race & National Origin
46	Position Occupied
47	Sex
48-49	Agency
50-51	Subelement
52-53	State/Country Code
54-55	Pay Plan
56-57	Grade
58-59	Step
60	Supervisor
61-62	Education Level
63	PATCO
64-65	GS Equivalent
66-67	TOA (Type of Appointment)
68	FEGLI
69	Citizenship
70	Retirement
71	Annuitant Indicator
72	VEV (Vietnam Era Veteran Ind.)

RECORD LAYOUT FOR FILE: CPDFFFX1 RECORD SIZE: 150 Ch
WORDS: 25

FILE DESCRIPTION: Master File

SORT SEQUENCE - Agency, SSN

CHARACTER POSITION	DATA
1-9	Social Security Number (SSN)
10-15	Service Computation Date(YYMMDD)
16-17	Retained Grade
18	Citizenship Code (YYMMDD)
19-24	Date of Birth (YYMMDD)
25	Work Schedule Code
26	Status Code
27-32	Separation Date (YYMMDD)
33-36	Submitting Office Number (SON)
37-42	Eff. Date Personnel Action
43-47	Salary
48-51	Occupation Code
52-53	Functional Classification
54-60	City, County, Geog. Location
61-64	SMSA
65-66	Special Program Identifier
67-68	Handicap Code
69	Filler (reserved field)
70	Pay Rate Determinate
71-72	Pay Basis Code
73	Veterans Preference Code
74	Tenure Code
75	Race & National Origin
76	FEGLI
77	Retirement Code
78	Position Occupied
79	PMIP
80	Sex
81-82	Agency Code
83-84	Subelement Code
85-86	State/County Geog. Location
87-88	Pay Plan
89-90	Pay Grade
91-92	Step (or Rate)
93-95	Nature of Action Code
96	Supervisory or Nonsup./Mgr.
97-98	Academic Educational Level
99-100	Year Degree Attained
101-104	Academic Discipline
105-106	Retained Step
107	PATCO
108-109	GS-Equivalent
110-111	Retained Pay Plan
112-115	Bargaining Unit Status
116	FLSA Exemption Status
117	Vietnam Era Veteran Indicator
118	Anuitant Indicator
119-121	Legal Authority (1)
122-124	Legal Authority (2)
125-127	Current Appointment Auth. (1)
128-130	Current Appointment Auth. (2)

RECORD LAYOUT FOR FILE: CPDFFFX1 RECORD SIZE: 150 Ch
WORDS: 25

FILE DESCRIPTION: Master File

SORT SEQUENCE - Agency, SSN

CHARACTER POSITION	DATA
131-132	Type of Appointment
133-136	Date Entered Cur. Grade (YYMM)
137-140	Date Entered Cur. Occup.(YYMM)
141-150	Filler

APPENDIX B - BENCHMARK QUERIES

APPENDIX B.1 - QUEL QUERY SETS

```
/*
/*   QUERY SET # 1
/*
/*   Single, small relation retrieval
/*
q1-1:  range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)

q1-2:  range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       where r.RET_PAY_PLAN = "WG"

q1-3:  range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       where (r.RET_PAY_PLAN = "WG"
          or  r.RET_PAY_PLAN = "GM")

q1-4:  range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       where (r.RET_PAY_PLAN = "WG"
          or  r.RET_PAY_PLAN = "GM")
         and  r.RET_GRADE > "08"

q1-5:  range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       where (r.RET_PAY_PLAN = "WG"
          or  r.RET_PAY_PLAN = "GM")
          and  r.RET_GRADE > "08"
          and  r.RET_GRADE < "12"

q1-6:  range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       where ((r.RET_PAY_PLAN = "WG"
           or  r.RET_PAY_PLAN = "GM")
           and r.RET_GRADE > "08"
           and   r.RET_GRADE < "12")
           or  r.RET_GRADE = "07"
/*
/*  QUERY SET # 2
/*
/*  Single, medium relation retrieval
/*
q2-1:  range of p is PERS_DATA
       retrieve (p.SSN, p.RACE, p.OCCUPATION)

q2-2:  range of p is PERS_DATA
       retrieve (p.SSN, p.RACE, p.OCCUPATION)
         where p.RACE = "C"

q2-3:  range of p is PERS_DATA
```

```
        retrieve (p.SSN, p.RACE, p.OCCUPATION)
          where (p.RACE = "C" or p.RACE = "L")

q2-4:   range of p is PERS_DATA
        retrieve (p.SSN, p.RACE, p.OCCUPATION)
          where (p.RACE = "C" or p.RACE = "L")
          and  p.SEX = "F"

q2-5:   range of p is PERS_DATA
        retrieve (p.SSN, p.RACE, p.OCCUPATION)
          where (p.RACE = "C" or p.RACE = "L")
          and p.SEX = "F"
          and p.BIRTH_DATE > 550101

/*
/*  QUERY SET # 3
/*
/*  Single, medium relation retrieval
/*
q3-1:   range of j is JOB_DETAIL
        retrieve (j.SSN, j.PAY_GRADE)

q3-2:   range of j is JOB_DETAIL
        retrieve (j.SSN, j.PAY_GRADE)
          where j.PATCO = "T"

q3-3:   range of j is JOB_DETAIL
        retrieve (j.SSN, j.PAY_GRADE)
          where (j.PATCO = "T" or j.PATCO = "O")

q3-4:   range of j is JOB_DETAIL
        retrieve (j.SSN, j.PAY_GRADE)
          where (j.PATCO = "T" or j.PATCO = "O")
          and j.PAY_GRADE < "11"

q3-5:   range of j is JOB_DETAIL
        retrieve (j.SSN, j.PAY_GRADE)
          where (j.PATCO = "T" or j.PATCO = "O")
          and j.PAY_GRADE < "11"
          and j.PAY_GRADE > "06"

q3-6:   range of j is JOB_DETAIL
        retrieve (j.SSN, j.PAY_GRADE)
          where (j.PATCO = "T" or j.PATCO = "O")
          and (j.PAY_GRADE < "11" and j.PAY_GRADE > "06")
          or j.BARG_UNIT = "0030"
/*
/*   QUERY SET # 4
/*
/*   Single, large relation retrieval
/*
```

```
q4-1:  range of a is AGENCY_DESC
       retrieve (a.AGENCY, a.SUBELEMENT)

q4-2:  range of a is AGENCY_DESC
       retrieve (a.AGENCY, a.SUBELEMENT)
         where (a.AGENCY = "BD" or a.AGENCY = "AF")

q4-3:  range of a is AGENCY_DESC
       retrieve (a.AGENCY, a.SUBELEMENT)
         where (a.AGENCY = "BD" or a.AGENCY = "AF")
         and a.SUBELEMENT = "07"

q4-4:  range of a is AGENCY_DESC
       retrieve (a.AGENCY, a.SUBELEMENT)
         where ((a.AGENCY = "BD" or a.AGENCY = "AF")
         and a.SUBELEMENT = "07")
         or a.SUBELEMENT = "24"
/*
/*   QUERY SET # 5
/*
/*   Single, large relation retrieval
/*
q5-1:  range of e is EDUCATION
       retrieve (e.SSN, e.EDUC_LEVEL)

q5-2:  range of e is EDUCATION
       retrieve (e.SSN, e.EDUC_LEVEL)
         where e.DEGREE_DATE > 80

q5-3:  range of e is EDUCATION
       retrieve (e.SSN, e.EDUC_LEVEL)
         where (e.DEGREE_DATE > 80 or e.DEGREE_DATE < 55)

q5-4:  range of e is EDUCATION
       retrieve (e.SSN, e.EDUC_LEVEL)
         where (e.DEGREE_DATE > 80 or e.DEGREE_DATE < 55)
         and e.EDUC_LEVEL > 13

q5-5:  range of e is EDUCATION
       retrieve (e.SSN, e.EDUC_LEVEL)
         where (e.DEGREE_DATE > 80 or e.DEGREE_DATE < 55)
         and e.EDUC_LEVEL > 13
         and e.ACAD_DISC = "0506"

q5-6:  range of e is EDUCATION
       retrieve (e.SSN, e.EDUC_LEVEL)
         where ((e.DEGREE_DATE > 81 or e.DEGREE_DATE < 55)
         and e.EDUC_LEVEL > 13
         and e.ACAD_DISC = "0506")
         or e.ACAD_DISC = "1701"
/*
```

```
/*   QUERY SET # 6
/*
/*   Two relation retrieval - small and medium relations
/*
q6-1:  range of r is RETAIN_DATA
       range of d is JOB_DETAIL
       retrieve (r.SSN, r.RET_GRADE, d.BARG_UNIT)
         where r.SSN = d.SSN

q6-2:  range of r is RETAIN_DATA
       range of d is JOB_DETAIL
       retrieve (r.SSN, r.RET_GRADE, d.BARG_UNIT)
         where r.SSN = d.SSN
         and j.PATCO = "T"

q6-3:  range of r is RETAIN_DATA
       range of d is JOB_DETAIL
       retrieve (r.SSN, r.RET_GRADE, d.BARG_UNIT)
         where r.SSN = d.SSN
         and (d.PATCO = "T" or d.PATCO = "O")

q6-4:  range of r is RETAIN_DATA
       range of d is JOB_DETAIL
       retrieve (r.SSN, r.RET_GRADE, d.BARG_UNIT)
         where r.SSN = d.SSN
         and ((d.PATCO = "T" or d.PATCO = "O")
         and  d.BARG_UNIT = "7777"
         and r.RET_GRADE < "08") or r.RET_PAY_PLAN = "WG"
/*
/*   QUERY SET # 7
/*
/*   Two relation retrieval, medium - medium
/*
q7-1:  range of m is PERS_MISC
       range of d is PERS_DATA
       retrieve (d.SSN, d.CITIZEN, m.VET_PREF)
       where d.SSN = m.SSN

q7-2:  range of m is PERS_MISC
       range of d is PERS_DATA
       retrieve (d.SSN, d.CITIZEN, m.VET_PREF)
         where d.SSN = m.SSN
         and d.HANDICAP = "01"

q7-3:  range of m is PERS_MISC
       range of d is PERS_DATA
       retrieve (d.SSN, d.CITIZEN, m.VET_PREF)
         where d.SSN = m.SSN
         and (d.HANDICAP = "01" or d.HANDICAP = "49")

q7-4:  range of m is PERS_MISC
```

```
        range of d is PERS_DATA
        retrieve (d.SSN, d.CITIZEN, m.VET_PREF)
          where d.SSN = m.SSN
          and (d.HANDICAP = "01" or d.HANDICAP = "49")
          and d.SEX = "M"

q7-5:   range of m is PERS_MISC
        range of d is PERS_DATA
        retrieve (d.SSN, d.CITIZEN, m.VET_PREF)
          where d.SSN = m.SSN
          and (d.HANDICAP = "01" or d.HANDICAP = "49")
          and d.SEX = "M"
          and m.VIET_VET = "V"

q7-6:   range of m is PERS_MISC
        range of d is PERS_DATA
        retrieve (d.SSN, d.CITIZEN, m.VET_PREF)
          where d.SSN = m.SSN
          and ((d.HANDICAP = "01" or d.HANDICAP = "49")
          and d.SEX = "M"
          and m.VIET_VET = "V")
          or m.VET_PREF = "*"
/*
/*   QUERY SET # 8
/*
/*   Two relation retrieval, large - large
/*
q8-1:   range of h is JOB_HISTORY
        range of e is EDUCATION
        retrieve (e.SSN, h.AGENCY, e.EDUC_LEVEL)
          where e.SSN = h.SSN

q8-2:   range of h is JOB_HISTORY
        range of e is EDUCATION
        retrieve (e.SSN, h.AGENCY, e.EDUC_LEVEL)
          where e.SSN = h.SSN
          and e.ACAD_DISC = "0506"

q8-3:   range of h is JOB_HISTORY
        range of e is EDUCATION
        retrieve (e.SSN, h.AGENCY, e.EDUC_LEVEL)
          where e.SSN = h.SSN
          and (e.ACAD_DISC = "0506" or e.ACAD_DISC = "0101")

q8-4:   range of h is JOB_HISTORY
        range of e is EDUCATION
        retrieve (e.SSN, h.AGENCY, e.EDUC_LEVEL)
          where e.SSN = h.SSN
          and (e.ACAD_DISC = "0506" or e.ACAD_DISC = "0101")
          and h.SERV_DATE > "780101"
```

```
q8-5:  range of h is JOB_HISTORY
       range of e is EDUCATION
       retrieve (e.SSN, h.AGENCY, e.EDUC_LEVEL)
         where e.SSN = h.SSN
         and (e.ACAD_DISC = "0506" or e.ACAD_DISC = "0101")
         and h.SERV_DATE > "780101"
         and (e.EDUC_LEVEL > "21" or h.STATE = "??")
/*
/*   QUERY SET # 9
/*
/*   Three relation retrieval
/*
q9-1:  range of p is PERS_DATA
       range of e is EDUCATION
       range of m is PERS_MISC
       retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
         where p.SSN = e.SSN and e.SSN = m.SSN

q9-2:  range of p is PERS_DATA
       range of e is EDUCATION
       range of m is PERS_MISC
       retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
         where p.SSN = e.SSN and e.SSN = m.SSN
         and (m.VET_PREF = "2")

q9-3:  range of p is PERS_DATA
       range of e is EDUCATION
       range of m is PERS_MISC
       retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
         where p.SSN = e.SSN and e.SSN = m.SSN
         and (m.VET_PREF = "2" or m.VET_PREF = "6")

q9-4:  range of p is PERS_DATA
       range of e is EDUCATION
       range of m is PERS_MISC
       retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
         where p.SSN = e.SSN and e.SSN = m.SSN
         and (m.VET_PREF = "2" or m.VET_PREF = "6")
         and  p.HANDICAP = "15"

q9-5:  range of p is PERS_DATA
       range of e is EDUCATION
       range of m is PERS_MISC
       retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
         where p.SSN = e.SSN and e.SSN = m.SSN
         and (m.VET_PREF = "2" or m.VET_PREF = "6")
         and p.HANDICAP = "15"
         and e.EDUC_LEVEL > "17"

q9-6:  range of p is PERS_DATA
       range of e is EDUCATION
```

```
      range of m is PERS_MISC
      retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
        where p.SSN = e.SSN and e.SSN = m.SSN
        and (m.VET_PREF = "2" or m.VET_PREF = "6")
        and p.HANDICAP = "15"
        and (e.EDUC_LEVEL > "17" or e.ACAD_DISC = "2209")
/*
/*   QUERY SET #10
/*
/*   Four relation retrieval
/*
q10-1: range of a is AGENCY_DESC
       range of h is JOB_HISTORY
       range of p is AUTH_PERS
       range of l is LEGAL_AUTH
       retrieve (a.AGENCY, p.SSN, h.SALARY, l.CODE, l.NAME)
         where a.AGENCY = h.AGENCY and h.SSN = p.SSN
         and p.CODE = l.CODE

q10-2: range of a is AGENCY_DESC
       range of h is JOB_HISTORY
       range of p is AUTH_PERS
       range of l is LEGAL_AUTH
       retrieve (a.AGENCY, p.SSN, h.SALARY, l.CODE, l.NAME)
         where a.AGENCY = h.AGENCY and h.SSN = p.SSN
         and p.CODE = l.CODE
         and a.AGENCY = "AF"

q10-3: range of a is AGENCY_DESC
       range of h is JOB_HISTORY
       range of p is AUTH_PERS
       range of l is LEGAL_AUTH
       retrieve (a.AGENCY, p.SSN, h.SALARY, l.CODE, l.NAME)
         where a.AGENCY = h.AGENCY and h.SSN = p.SSN
         and p.CODE = l.CODE
         and (a.AGENCY = "AF" or a.AGENCY = "BD")

q10-4: range of a is AGENCY_DESC
       range of h is JOB_HISTORY
       range of p is AUTH_PERS
       range of l is LEGAL_AUTH
       retrieve (a.AGENCY, p.SSN, h.SALARY, l.CODE, l.NAME)
         where a.AGENCY = h.AGENCY and h.SSN = p.SSN
         and p.CODE = l.CODE
         and (a.AGENCY = "AF" or a.AGENCY = "BD")
         and h.STATE = "31"

q10-5: range of a is AGENCY_DESC
       range of h is JOB_HISTORY
       range of p is AUTH_PERS
       range of l is LEGAL_AUTH
```

```
     retrieve (a.AGENCY, p.SSN, h.SALARY, l.CODE, l.NAME)
       where a.AGENCY = h.AGENCY and h.SSN = p.SSN
       and p.CODE = l.CODE
       and (a.AGENCY = "AF" or a.AGENCY = "BD")
       and (h.STATE = "31" or h.SALARY > 29000)
```

QUERY SETS WITH SORTING

```
/*
/*   QUERY SET # 1 with sorting
/*
/*   Single, small relation retrieval
/*
qo1-1: range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       order by RET_GRADE

qo1-2: range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       order by RET_GRADE
         where r.RET_PAY_PLAN = "WG"

qo1-3: range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       order by RET_GRADE
         where (r.RET_PAY_PLAN = "WG"
         or  r.RET_PAY_PLAN = "GM")

qo1-4: range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       order by RET_GRADE
         where (r.RET_PAY_PLAN = "WG"
         or  r.RET_PAY_PLAN = "GM")
         and  r.RET_GRADE > "08"

qo1-5: range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       order by RET_GRADE
         where (r.RET_PAY_PLAN = "WG"
         or  r.RET_PAY_PLAN = "GM")
         and  r.RET_GRADE > "08"
         and  r.RET_GRADE < "12"

qo1-6: range of r is RETAIN_DATA
       retrieve (r.SSN, r.RET_GRADE, r.RET_PAY_PLAN)
       order by RET_GRADE
         where ((r.RET_PAY_PLAN = "WG"
         or  r.RET_PAY_PLAN = "GM")
```

```
         and r.RET_GRADE > "08"
         and   r.RET_GRADE < "12")
         or  r.RET_GRADE = "07"
/*
/*  QUERY SET #2 with sorting
/*
/*  Single, medium relation retrieval
/*
qo2-1: range of p is PERS_DATA
       retrieve (p.SSN, p.RACE, p.OCCUPATION)
       order by OCCUPATION

qo2-2: range of p is PERS_DATA
       retrieve (p.SSN, p.RACE, p.OCCUPATION)
       order by OCCUPATION
         where p.RACE = "C"

qo2-3: range of p is PERS_DATA
       retrieve (p.SSN, p.RACE, p.OCCUPATION)
       order by OCCUPATION
         where (p.RACE = "C" or p.RACE = "L")

qo2-4: range of p is PERS_DATA
       retrieve (p.SSN, p.RACE, p.OCCUPATION)
       order by OCCUPATION
         where (p.RACE = "C" or p.RACE = "L")
         and  p.SEX = "F"

qo2-5: range of p is PERS_DATA
        retrieve (p.SSN, p.RACE, p.OCCUPATION)
        order by OCCUPATION
          where (p.RACE = "C" or p.RACE = "L")
          and p.SEX = "F"
          and p.BIRTH_DATE > 550101
/*
/*  QUERY SET # 3 with sorting
/*
/*  Single, medium relation retrieval
/*
qo3-1: range of j is JOB_DETAIL
       retrieve (j.SSN, j.PAY_GRADE)
       order by PAY_GRADE, SSN

qo3-2: range of j is JOB_DETAIL
       retrieve (j.SSN, j.PAY_GRADE)
       order by PAY_GRADE, SSN
         where j.PATCO = "T"

qo3-3: range of j is JOB_DETAIL
       retrieve (j.SSN, j.PAY_GRADE)
       order by PAY_GRADE, SSN
```

```
          where (j.PATCO = "T" or j.PATCO = "O")

qo3-4: range of j is JOB_DETAIL
       retrieve (j.SSN, j.PAY_GRADE)
       order by PAY_GRADE, SSN
         where (j.PATCO = "T" or j.PATCO = "O")
         and j.PAY_GRADE < "11"

qo3-5: range of j is JOB_DETAIL
       retrieve (j.SSN, j.PAY_GRADE)
       order by PAY_GRADE, SSN
         where (j.PATCO = "T" or j.PATCO = "O")
         and j.PAY_GRADE < "11"
         and j.PAY_GRADE > "06"

qo3-6: range of j is JOB_DETAIL
       retrieve (j.SSN, j.PAY_GRADE)
       order by PAY_GRADE, SSN
         where (j.PATCO = "T" or j.PATCO = "O")
         and (j.PAY_GRADE < "11" and j.PAY_GRADE > "06")
         or j.BARG_UNIT = "0030"
/*
/*   QUERY SET #4 with sorting
/*
/*   Single, large relation retrieval
/*
qo4-1: range of a is AGENCY_DESC
       retrieve (a.AGENCY, a,SUBELEMENT)
       order by SUBELEMENT

qo4-2: range of a is AGENCY_DESC
       retrieve (a.AGENCY, a.SUBELEMENT)
       order by SUBELEMENT
         where (a.AGENCY = "BD" or a.AGENCY = "AF")

qo4-3: range of a is AGENCY_DESC
       retrieve (a.AGENCY, a.SUBELEMENT)
       order by SUBELEMENT
         where (a.AGENCY = "BD" or a.AGENCY = "AF")
         and a.SUBELEMENT = "07"

qo4-4: range of a is AGENCY_DESC
       retrieve (a.AGENCY, a.SUBELEMENT)
       order by SUBELEMENT
         where ((a.AGENCY = "BD" or a.AGENCY = "AF")
         and a.SUBELEMENT = "07")
         or a.SUBELEMENT = "24"
```

QUERY SETS WITH AGGREGATION

```
/*
/*   QUERY SET # 4 with aggregation
/*
/*   Single, large relation retrieval
/*
qx4-1: range of a is AGENCY_DESC
       retrieve unique (a.SUBELEMENT,
         SUB_CNT = count(a.NAME by a.SUBELEMENT))

qx4-2: range of a is AGENCY_DESC
       retrieve unique (a.SUBELEMENT,
         SUB_CNT = count(a.NAME by a.SUBELEMENT))
         where (a.AGENCY = "BD" or a.AGENCY = "AF")

qx4-4: range of a is AGENCY_DESC
       retrieve unique (a.SUBELEMENT,
         SUB_CNT = count(a.NAME by a.SUBELEMENT))
         where (a.AGENCY = "BD" or a.AGENCY = "AF")
         and a.SUBELEMENT = "07"
/*
/*  QUERY SET #5 with aggregation
/*
/*  Single, large relation retrieval
/*
qx5-1: range of e is EDUCATION
       retrieve unique (a.EDUC_LEVEL,
         SSN_COUNT = count(e.SSN by e.EDUC_LEVEL))

qx5-2: range of e is EDUCATION
       retrieve unique (a.EDUC_LEVEL,
         SSN_COUNT = count(e.SSN by e.EDUC_LEVEL))
         where e.DEGREE_DATE > 80

qx5-3: range of e is EDUCATION
       retrieve unique (a.EDUC_LEVEL,
         SSN_COUNT = count(e.SSN by e.EDUC_LEVEL))
         where e.DEGREE_DATE > 80 or e.DEGREE_DATE < 55

qx5-4: range of e is EDUCATION
       retrieve unique (a.EDUC_LEVEL,
         SSN_COUNT = count(e.SSN by e.EDUC_LEVEL))
         where (e.DEGREE_DATE > 80 or e.DEGREE_DATE < 55)
         and e.EDUC_LEVEL > 13

qx5-5: range of e is EDUCATION
       retrieve unique (a.EDUC_LEVEL,
         SSN_COUNT = count(e.SSN by e.EDUC_LEVEL))
         where (e.DEGREE_DATE > 80 or e.DEGREE_DATE < 55)
         and e.EDUC_LEVEL > 13
         and e.ACAD_DISC = "0506"
```

```
qx5-6: range of e is EDUCATION
       retrieve unique (a.EDUC_LEVEL,
         SSN_COUNT = count(e.SSN by e.EDUC_LEVEL))
         where ((e.DEGREE_DATE > 80 or e.DEGREE_DATE < 55)
         and e.EDUC_LEVEL > 13
         and e.ACAD_DISC = "0506")
         or e.ACAD_DISC = "1701"
```

SPECIAL CASE QUERIES

```
/*
/* Special Case 1: Arrangement of Conditions
/*

sc1-1:  range of p is PERS_DATA
        range of e is EDUCATION
        range of m is PERS_MISC
        retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
          where p.SSN = e.SSN and e.SSN = m.SSN
                and m.SSN = 578608501

sc1-2:  range of p is PERS_DATA
        range of e is EDUCATION
        range of m is PERS_MISC
        retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
          where m.SSN = 578608501 and
                p.SSN = e.SSN and e.SSN = m.SSN

/*
/* Special Case 2: Implicit vs. Explicit Conditions
/*

sc2-1:  range of p is PERS_DATA
        range of e is EDUCATION
        range of m is PERS_MISC
        retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
          where p.SSN = e.SSN and e.SSN = m.SSN
                and m.SSN = 578608501

sc2-2:  range of p is PERS_DATA
        range of e is EDUCATION
        range of m is PERS_MISC
        retrieve (p.SSN, e.EDUC_LEVEL, p.BIRTH_DATE, m.VET_PREF)
          where p.SSN = 578608501 and e.SSN = 578608501
                and m.SSN = 578608501

/*
/* Special Case 3: Join Optimization
```

```
/*

sc3-1:  range of r is RETAIN_DATA
        range of d is JOB_DETAIL
        retrieve (r.SSN, r.RET_GRADE, d.BARG_UNIT)
          where r.SSN = d.SSN
                and (d.PATCO = "T" or d.PATCO = "O")

sc3-2:  range of r is RETAIN_DATA
        retrieve (r.SSN, r.RET_GRADE)
          order by SSN

sc3-3:  range of d is JOB_DETAIL
        retrieve into TEMP1 (d.SSN, d.BARG_UNIT)
          order by SSN
          where (d.PATCO = "T" or d.PATCO = "O")

sc3-4:  range of t is TEMP1
        retrieve (t.SSN, t.BARG_UNIT)
```

UPDATE QUERIES

```
/*
/* Insertions
/*

qI-1:  range of l is LEGAL_AUTH
       append to LEGAL_AUTH (
           CODE = "XXX",
           NAME = "NAME OF REGULATION")

qI-2:  range of e is EDUCATION
       append to EDUCATION (
           SSN = 155360283,
           EDUC_LEVEL = 18,
           DEGREE_DATE = 83,
           ACAD_DISC = "6X6X")

qI-3:  range of a is AGENCY_DESC
       append to AGENCY_DESC
           (AGENCY = "XX", SUBELEMENT = "YY",
            NAME = "NEW AGENCY", DESCRIPTION = "NEW DESCRIPTION")

/*
/* Deletions
/*
```

```
qD-1:   range of a is AGENCY_DESC
        delete a where a.AGENCY = "XX"

qD-2:   range of l is LEGAL_AUTH
        delete l where l.CODE = "XX"

qD-3:   range of e is EDUCATION
        delete e where e.SSN = 155360283

/*
/* Modifications
/*

qM-1:   range of l is LEGAL_AUTH
        replace l (NAME = "changed")
          where l.CODE = "Q7M"

qM-2:   range of e is EDUCATION
        replace e (DEGREE_DATE = 82)
          where e.SSN = 155360283
```

APPENDIX B.2 - SQL QUERY SETS

```
/*
/*  QUERY SET #1
/*
/*  Single, small relation query
/*
q1-1:  select ssn, ret_grade, ret_pay_plan
         from retain_data;
/*
q1-2:  select ssn, ret_grade, ret_pay_plan
         from retain_data;
         where ret_pay_plan = 'WG';
/*
q1-3:  select ssn, ret_grade, ret_pay_plan
         from retain_data;
         where ret_pay_plan = 'WG'
            or ret_pay_plan = 'GM';
/*
q1-4:  select ssn, ret_grade, ret_pay_plan
         from retain_data;
         where (ret_pay_plan = 'WG' or ret_pay_plan = 'GM')
            and ret_grade > '08';
/*
q1-5:  select ssn, ret_grade, ret_pay_plan
         from retain_data;
         where (ret_pay_plan = 'WG' or ret_pay_plan = 'GM')
            and ret_grade > '08'
            and ret_grade < '12';
/*
q1-6:  select ssn, ret_grade, ret_pay_plan
         from retain_data;
         where ((ret_pay_plan = 'WG' or ret_pay_plan = 'GM')
            and ret_grade > '08'
            and ret_grade < '12')
            or ret_grade = '07';
/*
/*  QUERY SET #2
/*
/*  Single, medium relation retrieval
/*
q2-1:  select ssn, race, occupation
         from pers_data;
/*
q2-2:  select ssn, race, occupation
         from pers_data
         where race = 'C';
/*
q2-3:  select ssn, race, occupation
         from pers_data
```

```
         where race = 'C' or race = 'L';
/*
q2-4:  select ssn, race, occupation
         from pers_data
         where (race = 'C' or race = 'L')
            and sex = 'F';
/*
q2-5:  select ssn, race, occupation
         from pers_data
         where (race = 'C' or race = 'L')
            and sex = 'F';
            and birth_date > 550101;
/*
/*  QUERY SET #3
/*
/*  Single, medium relation retrieval
/*
q3-1:  select ssn, pay_grade
         from job_detail;
/*
q3-2:  select ssn, pay_grade
         from job_detail
         where patco = 'T';
/*
q3-3:  select ssn, pay_grade
         from job_detail
         where patco = 'T' or patco = 'O';
/*
q3-4:  select ssn, pay_grade
         from job_detail
         where (patco = 'T' or patco = 'O')
            and pay_grade < '11';
/*
q3-5:  select ssn, pay_grade
         from job_detail
         where (patco = 'T' or patco = 'O')
            and pay_grade < '11'
            and pay_grade > '06';
/*
q3-6:  select ssn, pay_grade
         from job_detail
         where ((patco = 'T' or patco = 'O')
            and pay_grade < '11'
            and pay_grade > '06')
            or barg_unit = '0030';
/*
/*  QUERY SET #4
/*
/*  Single, large relation retrieval
/*
q4-1:  select agency, subelement
```

```
          from agency_desc;
/*
q4-2:  select agency, subelement
          from agency_desc
          where agency = ´BD´ or agency = ´AF´;
/*
q4-3:  select agency, subelement
          from agency_desc
          where (agency = ´BD´ or agency = ´AF´)
             and subelement = ´07´;
/*
q4-4:  select agency, subelement
          from agency_desc
          where ((agency = ´BD´ or agency = ´AF´)
             and subelement = ´07´)
             or subelement = ´24´;
/*
/*  QUERY SET #5
/*
q5-1:  select ssn, educ_level
          from education;
/*
q5-2:  select ssn, educ_level
          from education
          where degree_date > 80;
/*
q5-3:  select ssn, educ_level
          from education
          where degree_date > 80 or degree_date < 55;
/*
q5-4:  select ssn, educ_level
          from education
          where (degree_date > 80 or degree_date < 55)
             and educ_level > 13;
/*
q5-5:  select ssn, educ_level
          from education
          where (degree_date > 80 or degree_date < 55)
             and educ_level > 13
             and acad_disc = ´0506´;
/*
q5-6:  select ssn, educ_level
          from education
          where ((degree_date > 80 or degree_date < 55)
             and educ_level > 13
             and acad_disc = ´0506´)
             or acad_disc = ´1701´;
/*
/*  QUERY SET #6
/*
/*  Two relation retrieval - small and medium relations
```

```
/*
q6-1:  select retain_data.ssn, ret_grade, barg_unit
         from retain_data, job_detail
         where retain_data.ssn = job_detail.ssn;
/*
q6-2:  select retain_data.ssn, ret_grade, barg_unit
         from retain_data, job_detail
         where retain_data.ssn = job_detail.ssn
            and patco = 'T';
/*
q6-3:  select retain_data.ssn, ret_grade, barg_unit
         from retain_data, job_detail
         where retain_data.ssn = job_detail.ssn
            and (patco = 'T' or patco = 'O');
/*
q6-4:  select retain_data.ssn, ret_grade, barg_unit
         from retain_data, job_detail
         where retain_data.ssn = job_detail.ssn
            and (((patco = 'T' or patco = 'O')
            and barg_unit = '7777'
            and ret_grade < '08')
            or ret_pay_plan = 'WG');
/*
/*  QUERY SET #7
/*
/*  Two relation retrieval, medium - medium
/*
q7-1:  select pers_data.ssn, citizen, vet_pref
         from pers_data, pers_misc
         where pers_data.ssn = pers_misc.ssn;
/*
q7-2:  select pers_data.ssn, citizen, vet_pref
         from pers_data, pers_misc
         where pers_data.ssn = pers_misc.ssn
         and handicap = '01';
/*
q7-3:  select pers_data.ssn, citizen, vet_pref
         from pers_data, pers_misc
         where pers_data.ssn = pers_misc.ssn
         and handicap = '01' or handicap = '49';
/*
q7-4:  select pers_data.ssn, citizen, vet_pref
         from pers_data, pers_misc
         where pers_data.ssn = pers_misc.ssn
         and (handicap = '01' or handicap = '49')
         and sex = 'M';
/*
q7-5:  select pers_data.ssn, citizen, vet_pref
         from pers_data, pers_misc
         where pers_data.ssn = pers_misc.ssn
         and (handicap = '01' or handicap = '49')
```

```
         and sex = 'M'
         and viet_vet = 'V';
/*
q7-6:  select pers_data.ssn, citizen, vet_pref
         from pers_data, pers_misc
         where pers_data.ssn = pers_misc.ssn
         and (((handicap = '01' or handicap = '49')
         and sex = 'M'
         and viet_vet = 'V')
         or vet_pref like '%');
/*
/*  QUERY SET #8
/*
/*  Two relation retrieval, large - large
/*
q8-1:  select education.ssn, agency, educ_level
         from education, job_history
         where education.ssn = job_history.ssn;
/*
q8-2:  select education.ssn, agency, educ_level
         from education, job_history
         where education.ssn = job_history.ssn
            and acad_disc = '0506';
/*
q8-3:  select education.ssn, agency, educ_level
         from education, job_history
         where education.ssn = job_history.ssn
            and (acad_disc = '0506' or acad_disc = '0101');
/*
q8-4:  select education.ssn, agency, educ_level
         from education, job_history
         where education.ssn = job_history.ssn
            and (acad_disc = '0506' or acad_disc = '0101')
            and serv_date > 780101;
/*
q8-5:  select education.ssn, agency, educ_level
         from education, job_history
         where education.ssn = job_history.ssn
            and (((acad_disc = '0506' or acad_disc = '0101')
            and serv_date > 780101
            and educ_level > 21)
            or state = '31');
/*
/*  QUERY SET #9
/*
/*  Three relation retrieval
/*
q9-1:  select pers_data.ssn, educ_level, birth_date, vet_pref
         from pers_data, education, pers_misc
         where pers_data.ssn = education.ssn
            and education.ssn = pers_misc.ssn;
```

```
/*
q9-2:  select pers_data.ssn, educ_level, birth_date, vet_pref
         from pers_data, education, pers_misc
         where pers_data.ssn = education.ssn
            and education.ssn = pers_misc.ssn
            and vet_pref = '2';
/*
q9-3:  select pers_data.ssn, educ_level, birth_date, vet_pref
         from pers_data, education, pers_misc
         where pers_data.ssn = education.ssn
            and education.ssn = pers_misc.ssn
            and (vet_pref = '2' or vet_pref = '6');
/*
q9-4:  select pers_data.ssn, educ_level, birth_date, vet_pref
         from pers_data, education, pers_misc
         where pers_data.ssn = education.ssn
            and education.ssn = pers_misc.ssn
            and (vet_pref = '2' or vet_pref = '6')
            and handicap = '15';
/*
q9-5:  select pers_data.ssn, educ_level, birth_date, vet_pref
         from pers_data, education, pers_misc
         where pers_data.ssn = education.ssn
            and education.ssn = pers_misc.ssn
            and (vet_pref = '2' or vet_pref = '6')
            and handicap = '15'
            and educ_level > 17;
/*
q9-6:  select pers_data.ssn, educ_level, birth_date, vet_pref
         from pers_data, education, pers_misc
         where pers_data.ssn = education.ssn
            and education.ssn = pers_misc.ssn
            and (((vet_pref = '2' or vet_pref = '6')
            and handicap = '15'
            and educ_level > 17)
            or acad_disc = '2209');
/*
/*  QUERY SET #10
/*
/*  Four relation retrieval
/*
q10-1: select agency_desc.agency, auth_pers.ssn, job_history.salary,
legal_auth.code, legal_auth.name
         from agency_desc, job_history, auth_pers, legal_auth
         where agency_desc.agency = job_history.agency
         and job_history.ssn = auth_pers.ssn
         and auth_pers.code = legal_auth.code;
/*
q10-2: select agency_desc.agency, auth_pers.ssn, job_history.salary,
legal_auth.code, legal_auth.name
         from agency_desc, job_history, auth_pers, legal_auth
```

```
        where agency_desc.agency = job_history.agency
        and job_history.ssn = auth_pers.ssn
        and auth_pers.code = legal_auth.code
        and agency = 'AF';
/*
q10-3: select agency_desc.agency, auth_pers.ssn, job_history.salary,
legal_auth.code, legal_auth.name
        from agency_desc, job_history, auth_pers, legal_auth
        where agency_desc.agency = job_history.agency
        and job_history.ssn = auth_pers.ssn
        and auth_pers.code = legal_auth.code
        and agency = 'AF' or agency = 'BD';
/*
q10-4: select agency_desc.agency, auth_pers.ssn, job_history.salary,
legal_auth.code, legal_auth.name
        from agency_desc, job_history, auth_pers, legal_auth
        where agency_desc.agency = job_history.agency
        and job_history.ssn = auth_pers.ssn
        and auth_pers.code = legal_auth.code
        and (agency = 'AF' or agency = 'BD')
        and state = '31';
/*
q10-5: select agency_desc.agency, auth_pers.ssn, job_history.salary,
legal_auth.code, legal_auth.name
        from agency_desc, job_history, auth_pers, legal_auth
        where agency_desc.agency = job_history.agency
        and job_history.ssn = auth_pers.ssn
        and auth_pers.code = legal_auth.code
        and ((agency = 'AF' or agency = 'BD')
        and state = '31') or salary > 29000;
```

QUERY SETS WITH SORTING

```
/*
/*  QUERY SET #1 with sorting
/*
/*  Single, small relation retrieval
/*
qo1-1:  select ssn, ret_grade, ret_pay_plan
          from retain_data
          order by ret_grade;
/*
qo1-2:  select ssn, ret_grade, ret_pay_plan
          from retain_data
          where ret_pay_plan = 'WG'
          order by ret_grade;
/*
qo1-3:  select ssn, ret_grade, ret_pay_plan
```

```
          from retain_data
          where ret_pay_plan = ´WG´
             or ret_pay_plan = ´GM´
          order by ret_grade;
/*
qo1-4:  select ssn, ret_grade, ret_pay_plan
          from retain_data
          where (ret_pay_plan = ´WG´ or ret_pay_plan = ´GM´)
             and ret_grade > ´08´
          order by ret_grade;
/*
qo1-5:  select ssn, ret_grade, ret_pay_plan
          from retain_data
          where (ret_pay_plan = ´WG´ or ret_pay_plan = ´GM´)
             and ret_grade > ´08´
             and ret_grade < ´12´
          order by ret_grade;
/*
qo1-6:  select ssn, ret_grade, ret_pay_plan
          from retain_data
          where ((ret_pay_plan = ´WG´ or ret_pay_plan = ´GM´)
             and ret_grade > ´08´
             and ret_grade < ´12´)
             or ret_grade = ´07´
          order by ret_grade;
/*
/*  QUERY SET #2 with sorting
/*
/*  Single, medium relation retrieval
/*
qo2-1:  select ssn, race, occupation
          from pers_data
          order by occupation;
/*
qo2-2:  select ssn, race, occupation
          from pers_data
          where race = ´C´
          order by occupation;
/*
qo2-3:  select ssn, race, occupation
          from pers_data
          where race = ´C´ or race = ´L´
          order by occupation;
/*
qo2-4:  select ssn, race, occupation
          from pers_data
          where (race = ´C´ or race = ´L´)
             and sex = ´F´
          order by occupation;
/*
qo2-5:  select ssn, race, occupation
```

```
          from pers_data
          where (race = 'C' or race = 'L')
             and sex = 'F'
             and birth_date > 550101
          order by occupation;

/*
/*  QUERY SET #3 with sorting
/*
/*  Single, medium relation retrieval
/*
qo3-1:  select ssn, pay_grade
          from job_detail
          order by pay_grade, ssn;
/*
qo3-2:  select ssn, pay_grade
          from job_detail
          where patco = 'T'
          order by pay_grade, ssn;
/*
qo3-3:  select ssn, pay_grade
          from job_detail
          where patco = 'T' or patco = 'O'
          order by pay_grade, ssn;
/*
qo3-4:  select ssn, pay_grade
          from job_detail
          where (patco = 'T' or patco = 'O')
             and pay_grade < '11'
          order by pay_grade, ssn;
/*
qo3-5:  select ssn, pay_grade
          from job_detail
          where (patco = 'T' or patco = 'O')
             and pay_grade < '11'
             and pay_grade > '06'
          order by pay_grade, ssn;
/*
qo3-6:  select ssn, pay_grade
          from job_detail
          where ((patco = 'T' or patco = 'O')
             and pay_grade < '11'
             and pay_grade > '06')
             or barg_unit = '0030'
          order by pay_grade, ssn;
/*
/*  QUERY SET #4 with sorting
/*
/*  Single, large relation retrieval
/*
qo4-1:  select agency, subelement
```

```
          from agency_desc
          order by subelement;
/*
qo4-2:  select agency, subelement
          from agency_desc
          where agency = ´BD´ or agency = ´AF´
          order by subelement;
/*
qo4-3:  select agency, subelement
          from agency_desc
          where (agency = ´BD´ or agency = ´AF´)
             and subelement = ´07´
          order by subelement;
/*
qo4-4:  select agency, subelement
          from agency_desc
          where ((agency = ´BD´ or agency = ´AF´)
             and subelement = ´07´)
             or subelement = ´24´
          order by subelement;
```

QUERY SETS WITH AGGREGATION

```
/*
/*  QUERY SET #4 with aggregation
/*
/*  Single, large relation retrieval
/*
qx4-1:  select distinct subelement, count(name)
          from agency_desc
          group by subelement;
/*
qx4-2:  select distinct subelement, count(name)
          from agency_desc
          where (agency = ´BD´ or agency = ´AF´)
          group by subelement;
/*
qx4-3:  select distinct subelement, count(name)
          from agency_desc
          where (agency = ´BD´ or agency = ´AF´)
             and subelement = ´07´
          group by subelement;

/*
/*  QUERY SET #5 with aggregation
/*
/*  Single, large relation retrieval
/*
```

```
qx5-1:  select distinct educ_level, count(ssn)
          from education
          group by educ_level;
/*
qx5-2:  select distinct educ_level, count(ssn)
          from education
          where degree_date > 80
          group by educ_level;
/*
qx5-3:  select distinct educ_level, count(ssn)
          from education
          where degree_date > 80
             or degree_date < 55
          group by educ_level;
/*
qx5-4:  select distinct educ_level, count(ssn)
          from education
          where (degree_date > 80 or degree_date < 55)
             and educ_level > 13
          group by educ_level;
/*
qx5-5:  select distinct educ_level, count(ssn)
          from education
          where (degree_date > 80 or degree_date < 55)
             and educ_level > 13
             and acad_disc = '0506'
          group by educ_level;
/*
qx5-6:  select distinct educ_level, count(ssn)
          from education
          where ((degree_date > 80 or degree_date < 55)
             and educ_level > 13
             and acad_disc = '0506')
             or acad_disc = '1701'
          group by educ_level;
```

SPECIAL CASE QUERIES

```
/*
/* Special Case 1: Arrangement of Conditions
/*

scl-1:  select pers_data.ssn, educ_level, birth_date, vet_pref
          from pers_data, education, pers_misc
          where pers_data.ssn = education.ssn
          and education.ssn = pers_misc.ssn
          and pers_misc.ssn = 300378541
```

```
sc1-2:  select pers_data.ssn, educ_level, birth_date, vet_pref
          from pers_data, education, pers_misc
          where pers_misc.ssn = 300378541
          and pers_data.ssn = education.ssn
          and education.ssn = pers_misc.ssn

/*
/* Special Case 2: Implicit vs. Explicit Conditions
/*

sc2-1:  select pers_data.ssn, educ_level, birth_date, vet_pref
          from pers_data, education, pers_misc
          where pers_data.ssn = education.ssn
          and education.ssn = pers_misc.ssn
          and pers_misc.ssn = 300378541

sc2-2:  select pers_data.ssn, educ_level, birth_date, vet_pref
          from pers_data, education, pers_misc
          where pers_data.ssn = 300378541
          and education.ssn = 300378541
          and pers_misc.ssn = 300378541

/*
/* Special Case 3: Join Optimization
/*

sc3-1:  select retain_data.ssn, ret_grade, barg_unit
          from retain_data, job_detail
          where retain_data.ssn = job_detail.ssn
          and (patco = 'T' or patco = 'O')


sc3-2:  select ssn, ret_grade
          from retain_data
          order by ssn

sc3-3:  insert into templ
        select ssn, barg_unit
          from job_detail
          where (patco = 'T' or patco = 'O')

sc3-4:  select ssn, barg_unit
          from templ
```

UPDATE QUERIES

```
/*
/* Insertions
```

```
/*

qI-1:   insert into legal_auth (code, name)
          values ('XXX', 'NAME OF REGULATION')

qI-2:   insert into education (ssn, educ_level, degree_date, acad_disc)
          values (155360283, 18, 83, '6X6X')

qI-3:   insert into agency_desc (agency, subelement, name, description)
          values ('XX', 'YY', 'NEW AGENCY', 'NEW DESCRIPTION')

/*
/* Deletions
/*

qD-1:   delete from agency_desc
          where agency = 'XX'

qD-2:   delete from legal_auth
          where code = 'XX'

qD-3:   delete from education
          where ssn = 155360283

/*
/* Modifications
/*

qM-1:   update legal_auth
          set name = 'changed'
          where code = 'Q7M'

qM-2:   update education
          set degree_date = 82
          where ssn = 155360283
```

APPENDIX C - BENCHMARK DATA TABLES

APPENDIX C.1 - MICROCOMPUTER DATA TABLES

Table MICRO.1 - Result Size

Number of Records Retrieved					
q1-1 q1-2 q1-3 q1-4	74 46 47 35	q2-1 q2-2 q2-3 q2-4 q2-5	10,500 1,484 1,484 584 170	q3-1 q3-2 q3-3 q3-4 q3-5 q3-6	10,500 1,459 1,828 1,645 880 910
q4-1 q4-2 q4-3	10,500 10,500 10	q5-1 q5-2 q5-3 q5-4 q5-5 q5-6	11,152 135 9,754 629 18 70	q6-1 q6-2 q6-3 q6-4	74 7 7 46
q7-1 q7-2 q7-3 q7-4 q7-5 q7-6	10,500 764 871 674 142 10,500	q8-1 q8-2 q8-3 q8-4 q8-5	28,090 899 908 50 111	q9-1 q9-2 q9-3	141 43 45

Table MICRO.2 Response Time - Single Relation Queries

Query	Time to First: Index Level No Index	Time to First: Level 2	Time to First: Level 3	Time to Last: No Index	Time to Last: Level 1	Time to Last: Level 2	Time to Last: Level 3
1-1	2	3	3	11	11	11	11
1-2	2	2	3	11	11	11	10
1-3	2	3	4	11	11	12	11
1-4	3	3	4	12	12	11	11
2-1	2	3	3	377	385	391	394
2-2	3	8	9	219	230	87	82
2-3	3	10	12	224	237	238	189
2-4	3	11	11	193	200	203	128
2-5	4	*	*	140	144	141	*
3-1	3	4	4	245	235	238	249
3-2	4	10	11	132	121	64	66
3-3	3	12	14	138	130	136	138
3-4	3	10	*	140	131	105	*
3-5	3	8	*	105	106	86	*
3-6	3	*	*	113	114	114	*
4-1	2	3	3	216	226	223	231
4-2	3	12	13	228	233	235	248
4-3	4	7	7	87	88	11	11
5-1	2	3	3	222	217	218	220
5-2	4	4	6	84	83	83	17
5-3	2	11	12	216	207	207	211
5-4	3	12	*	206	200	201	*
5-5	4	11	*	92	94	93	*
5-6	3	*	98	100	97	*	

Response Time in Seconds

* Workspace Exceeded.

Table MICRO.3 Response Time - Single Relation Queries
Count Function

Query	Response Time in Seconds		
	Time to Last at Different Index Levels		
	No Index	Level 2	Level 3
q1-1	11	9	10
q1-2	9	10	11
q1-3	12	10	9
q1-4	10	10	12
q2-1	90	89	89
q2-2	94	93	48
q2-3	100	98	99
q2-4	100	101	76
q2-5	102	102	*
q3-1	79	78	79
q3-2	82	81	44
q3-3	89	87	86
q3-4	90	90	*
q3-5	96	94	*
q3-6	95	93	97
q4-1	69	69	70
q4-2	78	77	69
q4-3	86	86	10
q5-1	68	68	72
q5-2	73	73	14
q5-3	81	79	82
q5-4	86	84	*
q5-5	92	92	*
q5-6	98	98	98

* Workspace Exceeded

Table MICRO.4 Response Time - Multiple Relation Queries

Response Time in Seconds				
Query	Time to Last			
	Index Level No Index	Level 1	Level 2	Level 3
6-1	5,405	396	393	407
6-2	5,486	381	99	106
6-3	5,486	380	376	393
7-1	-	7,542	7,327	7,331
7-2	-	5,583	7,011	5540
7-3	-	5,659	4,970	5585
7-4	-	5,521	5,392	*
7-5	-	5,508	5,409	*
7-6	-	5,444	5,410	*
8-1	-	11,854	12,268	12,270
8-2	-	8,774	8,932	1,567
8-3	-	8,788	8,975	9,214
8-4	-	8,668	8,706	9,075
8-5	-	8,593	8,922	*
9-1	-	1,371	1,192	1291
9-2	-	577	482	488
9-3	-	597	496	497
9-4	-	387	324	*
9-5	-	359	17	*
9-6	-	371	301	*

* Workspace Exceeded

Table MICRO.5 Response Time - Update Queries

Response Time in Seconds				
Query	Index Level			
	No Index	Level 1	Level 2	Level 3
qI-1	5	5	6	10
qI-2	10	10	10	10
qI-3	9	10	11	12
qD-1	73	11	10	11
qD-2	9	9	11	9
qD-3	73	12	11	13
qM-1	10	10	9	9
qM-2	72	12	12	12

APPENDIX C.2 - MINICOMPUTER DATA TABLES

Table MINI.1 - Result Size - Single Relation Queries

Number of Records Retrieved			
Query	Database Size		
	3.5MB	6MB	10MB
1-1	74	129	163
1-2	46	72	78
1-3	47	73	81
1-4	35	50	55
1-5	14	18	21
1-6	18	28	33
2-1	10,500	20,000	33,000
2-2	1,484	1,989	2,774
2-3	1,484	2,088	2,874
2-4	584	875	1,223
2-5	170	257	359
3-1	10,500	20,000	33,000
3-2	1,459	2,668	7,618
3-3	1,828	3,408	8,563
3-4	1,645	3,095	8,078
3-5	880	1,636	3,615
3-6	910	1,667	3,646
4-1	10,500	20,000	33,000
4-2	10,500	19,948	19,948
4-3	10	23	23
4-4	10	23	23
5-1	11,152	21,349	35,534
5-2	135	227	421
5-3	9,754	18,921	28,997
5-4	630	1,298	1,645
5-5	18	30	34
5-6	70	117	159

Table MINI.2 - Response Time - Single Relation Queries
Level 1 Indexes

Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
1-1	380	360	440	1,690	2,690	3,350
1-2	340	530	360	1,470	2,210	2,750
1-3	330	310	310	1,530	2,510	2,880
1-4	350	490	380	1,480	2,250	2,750
1-5	380	540	400	1,350	1,520	2,520
1-6	450	390	450	1,510	2,250	2,840
2-1	360	980	380	229,000	437,190	721,490
2-2	440	740	470	169,240	80,920	510,690
2-3	440	440	560	182,710	340,910	567,250
2-4	1,900	1,940	1,910	173,140	323,900	539,500
2-5	2,130	2,190	2,120	172,250	321,080	534,480
3-1	340	370	360	195,940	373,370	613,230
3-2	370	720	440	149,200	100,650	486,150
3-3	390	390	440	159,820	302,290	517,210
3-4	390	680	430	160,840	492,120	519,760
3-5	460	570	520	154,000	251,010	485,620
3-6	520	500	580	164,580	310,580	514,430
4-1	330	980	350	184,640	351,310	574,770
4-2	400	310	380	191,040	363,210	517,440
4-3	25,260	530	25,200	115,820	11,730	364,440
4-4	27,450	27,460	27,470	125,640	238,820	394,210
5-1	350	1,000	410	203,330	391,010	642,230
5-2	1,750	1,730	1,760	131,970	251,670	418,850
5-3	360	340	390	205,510	394,170	631,830
5-4	520	900	510	137,360	177,850	432,450
5-5	5,780	1,140	5,760	132,330	217,370	418,820
5-6	3,740	3,760	3,800	137,790	260,980	439,540

Response Time in Milliseconds

Table MINI.3 - Response Time - Single Relation Queries
Level 2 Indexes

Response Time in Milliseconds						
Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
1-1	370	420	320	1,680	2,730	3,260
1-2	540	590	500	1,520	2,240	2,340
1-3	360	330	380	1,540	2,400	3,550
1-4	520	560	460	1,550	2,250	2,640
1-5	510	520	550	1,020	1,410	1,740
1-6	390	420	460	1,420	2,600	2,820
2-1	350	400	370	234,910	447,280	751,910
2-2	500	640	540	59,480	82,880	115,490
2-3	440	500	440	182,300	353,940	578,860
2-4	1,990	2,390	2,230	174,010	336,320	561,950
2-5	2,130	2,390	2,230	172,610	336,780	560,210
3-1	390	370	300	198,930	381,970	636,230
3-2	500	570	600	54,070	102,790	246,230
3-3	420	380	440	159,600	310,570	531,990
3-4	520	530	490	254,180	501,810	861,450
3-5	570	600	660	131,500	255,710	384,240
3-6	560	550	520	163,660	320,370	530,300
4-1	310	250	370	185,700	364,730	605,210
4-2	350	310	390	190,950	376,880	533,250
4-3	560	550	880	1,070	1,810	46,090
4-4	27,530	28,150	28,040	126,090	246,210	403,690
5-1	240	330	350	205,490	406,920	662,900
5-2	1,730	1,900	1,760	132,440	261,530	431,740
5-3	380	390	390	206,150	411,180	654,910
5-4	810	850	890	41,650	79,590	142,770
5-5	970	940	940	10,410	17,440	22,350
5-6	3,770	4,040	4,090	137,480	273,710	451,530

Table MINI.4 - Response Time - Single Relation Queries
No Indexes

Response Time in Milliseconds						
Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
1-1	380	300	---	1,690	2,890	---
1-2	390	350	---	1,540	2,390	---
1-3	350	310	---	1,550	2,440	---
1-4	350	380	---	1,480	2,360	---
1-5	410	500	---	1,390	2,330	---
1-6	440	450	---	1,490	2,330	---
2-1	380	340	---	230,840	452,030	---
2-2	460	490	---	168,730	324,220	---
2-3	560	550	---	182,750	354,170	---
2-4	1,960	2,110	---	172,700	338,430	---
2-5	2,120	2,100	---	172,900	341,270	---
3-1	410	340	---	196,080	381,400	---
3-2	400	380	---	148,930	288,820	---
3-3	420	350	---	159,500	311,620	---
3-4	410	430	---	159,950	312,790	---
3-5	470	480	---	154,190	298,840	---
3-6	540	500	---	164,400	318,050	---
4-1	370	330	---	183,980	362,320	---
4-2	370	310	---	190,180	373,820	---
4-3	25,300	26,260	---	115,740	225,210	---
4-4	27,420	28,500	---	125,860	246,220	---
5-1	380	300	---	205,650	402,370	---
5-2	1,850	1,820	---	132,350	259,040	---
5-3	310	400	---	205,510	407,110	---
5-4	490	490	---	137,140	270,610	---
5-5	5,770	5,790	---	132,590	261,430	---
5-6	3,790	3,770	---	137,640	272,790	---

Table MINI.5 - Response Time - Single Relation Queries
Level 1 Indexes - Buffer Effect Test

Query	Response Time in Milliseconds					
	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
1-1	310	320	440	1,630	2,830	3,630
1-2	250	270	250	1,380	2,200	2,560
1-3	220	260	240	1,370	2,250	2,650
1-4	250	290	230	1,340	2,410	2,500
1-5	280	340	280	1,200	1,950	2,320
1-6	290	380	340	1,330	2,160	2,750
2-1	320	250	400	230,650	455,030	736,360
2-2	520	570	510	170,070	324,860	525,990
2-3	490	510	550	182,780	351,050	580,470
2-4	1,960	1,910	1,870	174,160	336,310	551,390
2-5	2,180	2,170	2,350	172,400	333,020	547,040
3-1	350	290	480	196,010	382,870	628,820
3-2	410	370	380	148,800	290,380	498,250
3-3	390	370	430	159,520	311,670	528,270
3-4	460	420	400	160,170	312,190	533,270
3-5	460	400	500	153,750	300,160	495,230
3-6	500	530	500	164,420	319,770	524,550
4-1	310	250	460	184,150	363,570	591,840
4-2	250	290	370	190,490	373,210	530,080
4-3	25,170	25,640	25,720	115,730	234,680	372,950
4-4	27,440	28,060	28,490	125,850	245,840	405,680
5-1	330	270	450	204,610	401,210	656,750
5-2	1,800	1,790	1,850	132,320	259,940	430,660
5-3	370	390	370	205,340	404,070	653,450
5-4	490	510	470	136,970	271,390	444,960
5-5	5,670	5,780	5,730	132,150	260,090	428,620
5-6	3,760	3,940	3,920	137,600	268,690	449,990

Table MINI.6 - Response Time - Single Relation Queries
Level 1 Indexes - Sorted Results

Query	Response Time in Milliseconds					
	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
1-1	6,690	10,060	12,130	7,970	12,240	15,270
1-2	3,090	5,550	5,610	3,880	6,950	7,050
1-3	3,160	5,280	5,930	3,950	6,500	7,370
1-4	2,730	4,110	4,890	3,290	4,920	5,850
1-5	1,830	2,760	4,010	2,040	3,020	4,340
1-6	2,080	3,480	4,050	2,350	3,920	4,600
2-1	1,119,250	2,342,780	3,940,200	1,530,220	3,128,290	5,231,130
2-2	242,890	420,680	681,860	284,740	482,120	774,870
2-3	255,320	467,460	75780	300,340	533,110	852,250
2-4	200,580	375,080	615,730	211,260	391,720	647,210
2-5	178,530	343,730	568,990	181,420	347,930	576,130
3-1	801,130	1,606,020	2,836,540	1,089,880	2,165,500	3,827,200
3-2	219,630	426,070	926,920	253,340	492,520	1,133,730
3-3	248,480	500,330	1,017,170	293,860	588,160	1,253,570
3-4	241,380	472,520	991,450	280,250	547,790	1,207,570
3-5	195,600	377,970	691,060	211,960	411,650	774,300
3-6	210,430	405,120	727,360	228,160	440,120	811,270
4-1	791,530	1,572,680	2,640,570	1,054,190	2,083,950	3,504,900
4-2	787,450	1,608,760	1,774,260	1,051,590	2,115,560	2,298,350
4-3	116,220	227,730	375,870	116,360	228,080	376,230
4-4	126,800	245,670	409,160	126,950	246,020	409,570

Table MINI.7 - Response Time - Single Relation Queries
Level 2 Indexes - Sorted Results

Query	Response Time in Milliseconds					
	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
1-1	6,650	10,050	12,620	7,890	12,530	16,000
1-2	3,050	5,820	5,240	3,820	7,080	6,600
1-3	3,150	5,150	5,840	3,920	6,370	7,260
1-4	2,840	4,130	4,860	3,410	5,520	5,780
1-5	1,430	2,100	2,610	1,630	2,370	2,830
1-6	2,090	3,550	4,040	2,360	3,990	4,590
2-1	1,120,190	23,511,750	3,832,640	1,527,480	3,134,680	5,099,470
2-2	133,390	184,390	266,390	174,610	244,900	355,160
2-3	255,760	468,960	721,910	299,580	532,750	818,110
2-4	200,490	376,740	592,800	211,060	393,910	623,100
2-5	177,810	347,400	539,850	180,690	351,660	546,070
3-1	804,730	1,601,890	2,762,870	1,090,560	2,167,050	3,729,750
3-2	122,060	240,750	684,750	155,100	306,120	883,440
3-3	248,190	483,890	990,010	293,180	571,710	1,211,100
3-4	346,800	681,920	1,297,180	378,080	744,400	1,462,180
3-5	176,670	343,380	578,910	194,010	375,550	652,590
3-6	208,450	404,840	705,340	225,580	438,990	786,450
4-1	790,130	1,588,630	2,553,890	1,045,770	2,099,940	3,395,630
4-2	786,430	1,592,590	1,709,160	1,042,400	2,099,400	2,216,660
4-3	1,430	2,590	46,080	1,570	3,120	46,430
4-4	126,270	246,990	395,660	126,420	247,330	396,020

Table MINI.8 - Result Size - Single Relation Queries
Aggregate Queries

Number of Records Retrieved			
Query	Database Size		
	3.5MB	6MB	10MB
x4-1	36	45	68
x4-2	36	38	38
x4-3	1	1	1
x5-1	21	22	23
x5-2	7	8	8
x5-3	20	21	23
x5-4	7	8	10
x5-5	3	3	3
x5-6	6	6	7

Table MINI.9 - Response Time - Single Relation Queries
Level 1 Indexes - Aggregate Queries

Response Time in Milliseconds						
Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
x4-1	789,010	1,544,760	2,948,950	791,190	1,547,700	2,954,350
x4-2	794,960	1,549,710	1,725,050	797,140	1,552,360	1,727,580
x4-3	116,880	228,320	377,550	116,900	228,340	377,570
x5-1	904,150	1,752,710	2,965,480	905,290	1,754,210	2,966,770
x5-2	137,970	266,850	441,760	138,290	267,220	442,120
x5-3	773,330	1,517,290	2,389,150	774,440	1,518,470	2,390,420
x5-4	148,150	286,970	476,720	148,470	287,340	477,190
x5-5	133,360	259,920	428,770	133,480	260,040	428,890
x5-6	142,070	276,530	455,930	142,330	276,790	456,240

Table MINI.10 - Response Time - Single Relation Queries
Level 2 Indexes - Aggregate Queries

Response Time in Milliseconds						
Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
x4-1	787,210	1,528,440	2,832,860	789,560	1,531,460	2,836,890
x4-2	790,450	1,538,600	1,662,610	792,680	1,540,990	1,664,960
x4-3	1,200	1,860	45,020	1,220	1,880	45,040
x5-1	903,560	1,727,640	2,865,670	904,700	1,728,880	2,866,970
x5-2	138,970	261,690	430,220	139,290	262,060	430,580
x5-3	773,750	1,499,860	2,310,680	774,880	1,501,020	2,311,990
x5-4	37,090	68,190	126,460	37,400	68,560	126,950
x5-5	10,740	17,740	22,250	10,860	17,860	22,370
x5-6	141,600	268,780	445,960	141,870	269,040	446,270

Table MINI.11 - Result Size - Multiple Relation Queries

Number of Records Retrieved			
Query	Database Size		
	3.5MB	6MB	10MB
6-1	74	129	163
6-2	7	14	30
6-3	7	14	30
6-4	46	72	79
7-1	10,500	20,000	33,018
7-2	764	1,472	2,944
7-3	871	1,664	3,186
7-4	674	1,299	2,506
7-5	142	244	364
7-6	10,500	20,000	33,018
8-1	28,090	53,273	88,627
8-2	899	1,415	1,764
8-3	908	1,436	2,686
8-4	50	90	213
8-5	111	537	1,033
9-1	11,152	21,349	35,591
9-2	4,487	8,539	12,242
9-3	4,691	8,772	12,475
9-4	32	60	71
9-5	1	1	2
9-6	5	7	15

Table MINI.12 - Response Time - Multiple Relation Queries
Level 1 Indexes

Response Time in Milliseconds						
Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
6-1	2,740	2,970	2,920	377,840	821,220	1,348,580
6-2	12,980	13,440	13,620	137,310	347,740	654,420
6-3	24,700	27,740	28,050	378,470	824,660	1,358,270
6-4	2,820	3,130	3,180	380,390	826,770	1,355,400
7-1	1,900	970	700	668,120	1,268,230	3,138,500
7-2	880	800	840	611,390	1,153,960	2,973,060
7-3	850	800	830	614,590	1,174,060	2,975,690
7-4	930	810	860	616,980	1,174,930	2,986,760
7-5	2,190	2,240	2,740	273,980	494,730	921,860
7-6	910	840	840	677,130	1,294,530	3,193,810
8-1	420	2,160	2,060	2,508,050	4,767,470	8,416,560
8-2	4,230	4,240	4,730	2,291,770	4,323,700	7,729,520
8-3	4,280	4,250	4,800	2,309,650	4,360,630	7,837,480
8-4	9,300	9,340	11,170	564,230	1,066,680	2,177,960
8-5	10,550	10,490	11,660	2,337,270	4,430,410	7,880,420
9-1	1,740	1,220	1,160	1,473,720	2,862,690	5,884,110
9-2	1,090	1,090	1,170	732,820	1,384,750	2,611,040
9-3	1,180	1,140	1,180	1,433,920	2,791,760	5,744,600
9-4	9,390	9,130	12,690	1,386,630	2,691,640	5,621,160
9-5	926,370	935,060	972,060	936,530	1,822,500	3,116,780
9-6	9,800	9,790	13,240	1,398,420	2,717,990	5,752,380

Table MINI.13 - Response Time - Multiple Relation Queries
Level 2 Indexes

Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
6-1	2,590	2,900	4,250	384,780	822,110	1,339,890
6-2	5,940	6,690	7,160	77,780	158,610	399,670
6-3	25,000	27,590	27,790	386,260	824,410	1,344,760
6-4	2,830	3,320	3,130	391,810	825,400	1,348,590
7-1	770	850	1,820	679,530	1,264,950	3,131,000
7-2	970	1,120	1,260	108,900	180,040	457,830
7-3	840	740	800	628,960	1,170,800	2,963,660
7-4	890	860	930	631,820	1,173,240	2,972,150
7-5	2,300	2,070	2,720	277,620	493,710	921,130
7-6	950	880	860	690,360	1,293,580	3,189,690
8-1	700	610	730	2,480,710	4,710,160	8,639,260
8-2	970	1,050	1,320	99,790	161,210	223,200
8-3	4,520	4,440	5,240	2,296,540	4,349,240	8,017,500
8-4	9,270	9,310	11,370	693,040	1,064,520	2,207,100
8-5	10,400	10,310	12,180	2,309,670	4,422,050	8,100,380
9-1	1,050	1,070	1,840	1,503,570	2,845,860	5,873,570
9-2	1,330	1,150	1,520	697,560	1,289,090	2,430,920
9-3	1,160	1,280	1,220	1,463,320	3,276,790	5,730,840
9-4	1,440	1,450	1,600	28,880	45,440	76,190
9-5	17,510	15,550	19,040	17,860	29,330	46,370
9-6	9,950	9,590	13,230	1,427,180	2,717,030	5,632,630

Response Time in Milliseconds

Table MINI.14 - Response Time - Update Queries

Response Time in Milliseconds				
Query	Index Level	Time to Completion		
		3.5MB	6MB	10MB
I-3	No 1 2	--- 1,270 530	--- 420 750	--- 840 500
I-2	No 1 2	--- 500 840	--- 400 990	--- 630 1,130
I-3	No 1 2	--- 580 570	--- 520 590	--- 740 680
D-1	No 1 2	--- 320 370	--- 280 350	--- 310 310
D-2	No 1 2	--- 280 410	--- 330 380	--- 300 360
D-3	No 1 2	--- 380 480	--- 360 510	--- 300 460
M-1	No 1 2	--- 40,090 39,260	--- 74,550 75,540	--- 183,970 125,540
M-2	No 1 2	--- 430 440	--- 490 470	--- 540 550

APPENDIX C.3 - DATABASE MACHINE (DBM) DATA TABLES

Table DBM.1 - Result Size - Single Relation Queries

Number of Records Retrieved					
Query	Database Size				
	3.5MB	6MB	8MB	10MB	56MB
1-1	74	129	145	163	708
1-2	46	72	73	78	228
1-3	47	73	75	81	258
1-4	35	50	52	55	180
1-5	14	18	19	21	105
1-6	18	28	31	33	174
2-1	10,500	20,000	25,500	33,000	189,960
2-2	1,484	1,989	2,438	2,774	28,739
2-3	1,484	2,088	2,537	2,874	29,474
2-4	584	875	1,080	1,223	15,741
2-5	170	257	308	359	4,274
3-1	10,500	20,000	25,500	33,000	189,960
3-2	1,459	2,668	4,604	7,618	31,471
3-3	1,828	3,408	5,446	8,563	37,937
3-4	1,645	3,095	5,061	8,078	33,842
3-5	880	1,636	2,467	3,615	14,707
3-6	910	1,667	2,498	3,646	15,503
4-1	10,500	20,000	25,500	33,000	189,960
4-2	10,500	19,948	19.948	19.948	19.974
4-3	10	23	23	23	23
5-1	11,152	21,349	27,363	35,534	201,925
5-2	135	227	309	421	1,398
5-3	9,754	18,921	23,026	28,997	161,088
5-4	629	1,296	1,451	1,636	9,267
5-5	18	30	32	34	155
5-6	70	117	135	159	1,122

Table DBM.2 - Response Time - Single Relation Queries
Level 1 Indexes

Time to First Record in Milliseconds					
Query	Database Size				
	3.5MB	6MB	8MB	10MB	56MB
1-1	1,117	834	817	850	917
1-2	1,083	1,016	1,033	1,083	1,117
1-3	1,000	1,084	933	934	1,183
1-4	1,317	1,317	1,233	1,184	1,266
1-5	1,350	1,417	1,333	1,367	1,400
1-6	1,333	1,333	1,250	1,800	1,366
2-1	816	783	733	716	916
2-2	1,400	1,017	867	867	1,450
2-3	1,100	1,050	983	966	1,200
2-4	2,300	1,500	1,300	1,333	2,317
2-5	3,050	2,283	2,433	2,617	6,100
3-1	916	833	867	817	934
3-2	1,484	1,100	1,117	1,050	1,334
3-3	1,250	1,134	1,184	1,117	1,267
3-4	1,534	1,150	1,300	1,100	1,500
3-5	2,400	1,950	1,433	1,350	1,500
3-6	2,116	1,917	1,433	1,350	1,500
4-1	933	950	900	850	950
4-2	1,050	1,100	1,017	1,050	1,117
4-3	7,833	13,083	16,567	21,200	127,550
5-1	733	950	867	784	950
5-2	2,300	1,767	1,600	1,567	2,100
5-3	950	1,134	1,050	917	1,117
5-4	1,450	1,200	1,134	1,083	1,333
5-5	8,283	11,817	6,483	6,400	8,333
5-6	4,383	4,500	3,900	3,983	4,183

Table DBM.2 (cont.) - Response Time - Single Relation Queries
Level 1 Indexes

Time to Last Record in Milliseconds					
Query	Database Size				
	3.5MB	6MB	8MB	10MB	56MB
1-1	2,100	2,617	3,034	3,133	11,517
1-2	1,633	1,916	1,950	2,067	4,333
1-3	1,566	2,000	1,867	1,950	4,933
1-4	1,633	1,900	1,850	1,834	3,733
1-5	1,367	1,433	1,350	1,534	2,784
1-6	1,333	1,550	1,500	2,084	3,783
2-1	193,366	360,233	461,067	591,633	3,406,483
2-2	29,666	45,400	45,517	53,467	523,317
2-3	30,817	49,350	51,033	61,950	551,200
2-4	22,083	39,317	31,200	39,733	330,584
2-5	21,734	38,833	26,833	34,584	207,600
3-1	111,183	210,100	269,233	344,650	1,982,084
3-2	16,950	29,183	48,850	80,234	330,050
3-3	20,500	36,467	58,117	90,067	396,817
3-4	19,367	34,200	54,134	85,317	355,334
3-5	17,517	31,917	31,316	43,134	210,266
3-6	18,850	35,083	37,766	49,850	271,650
4-1	102,500	192,600	246,817	317,966	1,825,764
4-2	102,317	192,517	197,000	204,166	353,233
4-3	7,850	13,100	16,584	21,217	127,583
5-1	173,233	322,417	416,750	538,434	3,047,616
5-2	8,733	15,300	12,383	16,350	98,367
5-3	149,433	286,300	348,817	437,584	2,432,450
5-4	10,383	20,384	23,050	27,150	169,383
5-5	8,283	14,850	10,950	14,350	85,066
5-6	11,616	21,917	23,734	30,867	175,466

Table DBM.3 - Response Time - Single Relation Queries
Level 2 Indexes

Time to First Record in Milliseconds				
Query	Database Size			
	3.5MB	6MB	10MB	56MB
1-1	1,267	866	900	983
1-2	1,867	1,417	1,150	1,767
1-3	1,184	1,134	967	1,166
1-4	2,533	2,084	1,734	3,600
1-5	3,083	2,484	1,900	3,600
1-6	1,333	1,334	1,266	1,450
2-1	867	766	750	983
2-2	1,816	1,450	1,117	2,284
2-3	1,116	1,084	983	1,184
2-4	2,334	1,850	1,333	1,733
2-5	3,067	2,633	2,684	4,334
3-1	967	850	817	1,084
3-2	2,333	1,533	1,384	2,333
3-3	1,300	1,150	1,150	1,316
3-4	2,467	2,533	2,500	10,317
3-5	12,116	18,883	16,900	98,433
3-6	2,034	1,850	1,384	1,817
4-1	933	950	816	1,017
4-2	1,084	1,133	983	1,166
4-3	1,566	1,650	8,400	70,050
5-1	800	984	816	983
5-2	2,234	1,933	1,583	2,450
5-3	966	1,183	966	1,167
5-4	7,583	11,850	10,817	81,700
5-5	5,317	6,700	3,850	11,767
5-6	4,400	4,534	4,034	4,266

Table DBM.3 (cont.) - Response Time - Single Relation Queries
Level 2 Indexes

Query	Time to Last Record in Milliseconds Database Size			
	3.5MB	6MB	10MB	56MB
1-1	2,333	2,633	3,166	11,816
1-2	2,817	2,317	2,133	9,484
1-3	1,734	2,050	1,983	4,883
1-4	3,150	2,667	2,384	6,084
1-5	3,100	2,500	2,033	4,950
1-6	1,350	1,550	1,550	3,850
2-1	189,733	359,150	599,700	3,044,833
2-2	29,133	37,533	51,434	543,317
2-3	29,033	49,617	61,983	551,284
2-4	22,034	42,650	39,716	329,133
2-5	21,667	42,450	34,817	200,850
3-1	111,234	209,000	345,983	1,981,667
3-2	18,166	30,250	81,234	349,283
3-3	20,350	36,817	90,133	397,283
3-4	93,967	183,833	191,467	1,089,083
3-5	61,850	115,700	107,433	580,267
3-6	18,800	35,566	49,850	273,783
4-1	103,300	193,333	319,533	1,825,267
4-2	102,950	193,566	204,200	352,216
4-3	1,583	1,667	8,400	70,067
5-1	169,934	322,600	547,933	3,046,200
5-2	8,617	14,433	16,366	98,917
5-3	147,833	286,417	441,266	2,431,333
5-4	18,133	42,317	45,867	324,200
5-5	5,333	7,683	6,100	36,817
5-6	11,583	21,200	31,017	175,366

Table DBM.4 - Response Time - Single Relation Queries
No Indexes

Query	Response Time in Milliseconds	
	Time to First	Time to Last
	6MB	6MB
1-1	1,167	2,917
1-2	867	1,800
1-3	950	1,866
1-4	1,117	1,700
1-5	1,167	1,167
1-6	1,234	1,450
2-1	717	359,983
2-2	900	38,517
2-3	983	42,816
2-4	1,350	25,384
2-5	2,300	20,100
3-1	817	210,333
3-2	1,000	28,817
3-3	1,100	36,567
3-4	1,234	33,467
3-5	1,283	22,366
3-6	1,400	28,733
4-1	850	195,033
4-2	1,100	194,633
4-3	13,300	13,516
5-1	950	325,117
5-2	1,583	9,916
5-3	1,200	288,117
5-4	1,400	20,483
5-5	6,633	8,550
5-6	4,233	18,683

Table DBM.5 - Response Time - Single Relation Queries
Level 1 Indexes - Buffer Effect Test

Time to First Record in Milliseconds					
Query	Database Size				
	3.5MB	6MB	8MB	10MB	56MB
1-1	800	1,033	783	1,117	633
1-2	884	917	817	900	683
1-3	900	900	866	883	767
1-4	1,017	1,000	983	1,017	883
1-5	1,100	1,100	1,067	1,133	966
1-6	1,217	1,816	1,150	1,167	933
2-1	767	716	717	784	650
2-2	1,667	1,300	1,100	1,150	1,167
2-3	1,367	1,434	1,167	1,233	1,216
2-4	2,550	1,983	1,567	1,650	1,634
2-5	3,317	2,534	2,517	2,900	4,500
3-1	800	700	684	667	800
3-2	1,517	1,116	1,150	1,084	1,150
3-3	1,234	1,134	1,200	1,150	1,217
3-4	1,683	1,300	1,333	1,284	1,350
3-5	2,616	2,150	1,550	1,450	1,833
3-6	2,150	1,900	1,500	1,500	1,600
4-1	817	750	667	866	747
4-2	850	867	800	867	900
4-3	4,200	13,183	16,617	21,300	127,383
5-1	750	933	883	800	817
5-2	2,733	2,400	1,950	1,817	2,017
5-3	833	883	833	783	750
5-4	1,600	1,467	1,334	1,133	1,233
5-5	9,350	13,050	7,266	7,050	8,550
5-6	9,167	5,000	4,266	4,133	4,067

Table DBM.5 (cont.) - Response Time - Single Relation Queries
Level 1 Indexes - Buffer Effect Test

Time to Last Record in Milliseconds					
Query	Database Size				
	3.5MB	6MB	8MB	10MB	56MB
1-1	1,234	2,783	2,866	3,400	10,617
1-2	1,417	1,817	1,750	1,950	3,283
1-3	1,450	1,850	1,833	1,933	3,750
1-4	1,334	1,600	1,600	1,667	2,483
1-5	1,100	1,116	1,083	1,283	1,367
1-6	1,250	2,033	1,400	1,567	2,400
2-1	189,367	374,816	458,800	592,117	3,461,867
2-2	29,167	46,050	45,534	53,950	527,050
2-3	30,417	51,134	51,350	62,350	559,750
2-4	22,483	39,917	31,283	40,233	330,850
2-5	21,983	39,134	26,884	35,400	199,434
3-1	109,983	210,517	266,584	344,867	1,981,316
3-2	16,983	29,433	48,934	80,550	332,350
3-3	20,134	37,134	57,734	90,133	395,766
3-4	19,133	34,966	54,066	85,517	358,083
3-5	18,033	32,483	31,734	43,500	215,633
3-6	19,083	34,817	38,466	50,383	271,167
4-1	101,667	194,983	246,000	351,166	1,826,150
4-2	101,566	193,317	196,117	203,550	331,500
4-3	4,200	13,200	16,634	21,317	
5-1	168,500	331,866	413,650	536,700	3,066,500
5-2	9,317	16,134	14,017	17,617	96,283
5-3	147,550	294,583	350,633	437,366	2,442,633
5-4	10,683	21,150	23,450	27,100	168,150
5-5	9,366	16,067	11,850	14,750	75,517
5-6	12,550	22,650	25,100	32,050	172,083

Table DBM.6 - Response Time - Single Relation Queries
Level 1 Indexes - Sorted Result

Time to First Record in Milliseconds					
Query	Database Size				
	3.5MB	6MB	8MB	10MB	56MB
o1-1	1,284	1,650	1,566	1,700	5,984
o1-2	1,317	1,367	1,467	1,500	2,850
o1-3	1,216	1,333	1,317	1,417	2,867
o1-4	1,467	1,600	1,434	1,483	2,816
o1-5	1,550	1,400	1,283	1,417	2,566
o1-6	1,400	1,550	1,400	1,550	2,850
o2-1	130,750	270,367	349,317	465,516	3,506,084
o2-2	32,017	52,000	44,217	56,700	562,800
o2-3	36,200	59,284	56,166	72,116	640,467
o2-4	25,067	44,633	36,984	46,683	408,300
o2-5	22,384	39,983	27,417	36,000	234,083
o3-1	123,166	258,400	331,267	463,317	3,264,717
o3-2	23,967	44,300	63,566	108,567	576,750
o3-3	31,267	58,033	81,917	128,283	709,584
o3-4	28,917	56,200	76,900	124,200	667,550
o3-5	22,134	42,367	46,733	64,884	375,084
o3-6	25,433	46,450	55,200	80,350	444,733
o4-1	97,516	209,783	273,534	393,784	2,654,883
o4-2	102,267	217,900	223,784	230,917	380,666
o4-3	7,850	13,300	16,433	124,200	127,850

Table DBM.6 (cont.) - Response Time - Single Relation Queries
Level 1 Indexes - Sorted Result

Time to Last Record in Milliseconds				
Query	Database Size			
	3.5MB	6MB	8MB	10MB
o1-1	2,217	3,467	3,583	3,966
o1-2	1,883	2,287	2,384	2,483
o1-3	1,783	2,266	2,250	2,450
o1-4	1,784	2,300	2,084	2,133
o1-5	1,567	1,400	1,300	1,550
o1-6	1,416	1,766	1,650	1,833
o2-1	319,000	631,033	820,850	1,080,216
o2-2	59,200	87,417	88,133	106,983
o2-3	64,350	98,017	101,800	124,216
o2-4	35,543	63,417	56,350	68,816
o2-5	25,234	45,316	32,700	42,500
o3-1	233,333	470,466	607,950	818,450
o3-2	39,433	72,216	111,433	188,383
o3-3	51,533	94,366	138,567	217,367
o3-4	46,467	92,300	129,517	209,300
o3-5	32,550	59,217	72,250	104,850
o3-6	34,850	63,850	81,050	119,250
o4-1	198,916	401,683	531,034	722,317
o4-2	207,450	416,050	415,767	422,534
o4-3	7,866	13,300	16,450	21,183

Table DBM.7 - Result Size - Single Relation Queries
Aggregate Queries

Number of Records Retrieved			
Query	Database Size		
	3.5MB	6MB	10MB
x4-1	36	45	68
x4-2	36	38	38
x4-3	1	1	1
x5-1	21	22	23
x5-2	7	8	8
x5-3	20	21	23
x5-4	7	8	10
x5-5	3	3	3
x5-6	6	6	7

Table DBM.8 - Response Time - Single Relation Queries
Level 1 Indexes - Aggregate Queries

Response Time in Milliseconds						
Query	Time to First			Time to Last		
	3.5MB	6MB	10MB	3.5MB	6MB	10MB
x4-1	42,500	83,616	138,967	42,700	83,883	139,500
x4-2	219,150	464,600	680,483	219,333	464,817	680,683
x4-3	44,216	96,450	166,616	44,216	96,450	166,616
x5-1	45,817	87,366	150,450	45,817	87,383	150,467
x5-2	54,700	104,050	159,733	54,733	104,050	159,750
x5-3	193,533	381,266	603,134	193,550	381,266	603,150
x5-4	64,733	123,716	190,967	64,750	123,716	190,967
x5-5	52,600	99,833	155,384	52,660	99,833	155,384
x5-6	121,017	233,050	374,783	121,017	233,050	374,800

Table DBM.9 - Result Size - Multiple Relation Queries

Number of Records Retrieved					
Query	Database Size				
	3.5MB	6MB	8MB	10MB	56MB
6-1	74	129	145	163	---
6-2	7	14	22	30	97
6-3	7	14	22	30	111
6-4	46	72	74	79	---
7-1	10,500	20,000	---	---	---
7-2	764	1,472	---	---	---
7-3	871	1,664	---	---	---
7-4	674	1,299	1,890	---	---
7-5	142	244	---	364	---
7-6	10,500	20,000	---	---	---
8-1	28,090	53,273	---	---	---
8-2	899	1,415	---	---	---
8-3	908	1,436	---	---	---
8-4	50	90	---	---	---
8-5	111	537	---	---	---
9-1	11,152	21,349	---	---	---
9-2	4,487	8,539	---	---	---
9-3	4,691	8,772	---	---	---
9-4	32	60	---	---	---
9-5	1	1	---	---	---
9-6	5	7	---	---	---

Table DBM.10 - Response Time - Multiple Relation Queries
Level 1 Indexes

Time to First Record in Milliseconds					
Query	Database Size				
	3.5MB	6MB	8MB	10MB	56MB
6-1	1,784	1,484	77,184	77,450	-----
6-2	3,383	3,967	99,633	98,867	108,900
6-3	2,233	3,417	234,816	236,800	253,816
6-4	1,800	2,083	107,250	107,284	-----
7-1	1,433	1,367	-----	-----	-----
7-2	4,583	4,033	-----	-----	-----
7-3	9,233	9,083	-----	-----	-----
7-4	7,683	7,600	206,200	-----	-----
7-5	16,784	14,700	-----	1,319,767	-----
7-6	1,684	1,833	-----	-----	-----
8-1	1,433	1,300	-----	-----	-----
8-2	1,666	1,500	-----	-----	-----
8-3	2,717	2,934	-----	-----	-----
8-4	30,434	29,184	-----	-----	-----
8-5	37,650	38,833	-----	-----	-----
9-1	2,034	1,750	-----	-----	-----
9-2	2,150	1,967	-----	-----	-----
9-3	3,767	3,750	-----	-----	-----
9-4	14,700	13,450	-----	-----	-----
9-5	25,350	45,267	-----	-----	-----
9-6	559,016	1,085,000	-----	-----	-----

Table DBM.10 (cont.) - Response Time - Multiple Relation Queries
Level 1 Indexes

Time to Last Record in Milliseconds					
Query	Database Size				
	3.5MB	6MB	8MB	10MB	56MB
6-1	3,467	4,984	524,467	687,250	-----
6-2	3,400	3,984	112,116	178,783	-----
6-3	2,233	3,433	254,183	357,133	-----
6-4	2,750	4,233	626,517	822,550	-----
7-1	169,083	319,384	-----	-----	-----
7-2	50,050	83,850	-----	-----	-----
7-3	180,416	344,483	-----	-----	-----
7-4	95,216	188,850	15,995,783	-----	-----
7-5	62,100	99,800	-----	27,405,117	-----
7-6	216,734	411,717	-----	-----	-----
8-1	522,033	964,933	-----	-----	-----
8-2	28,533	41,117	-----	-----	-----
8-3	178,083	342,000	-----	-----	-----
8-4	118,234	216,967	-----	-----	-----
8-5	345,500	663,933	-----	-----	-----
9-1	528,734	1,028,567	-----	-----	-----
9-2	209,000	378,817	-----	-----	-----
9-3	545,284	1,059,783	-----	-----	-----
9-4	32,217	55,067	-----	-----	-----
9-5	25,350	45,284	-----	-----	-----
9-6	559,033	1,085,016	-----	-----	-----

Table DBM.11 - Response Time - Multiple Relation Queries
Level 2 Indexes

Time to First Record in Milliseconds				
Query	Database Size			
	3.5MB	6MB	10MB	56MB
6-1	1,700	1,484	77,467	-----
6-2	2,300	3,917	101,600	148,884
6-3	2,150	3,267	237,033	254,400
6-4	1,766	1,750	107,267	-----
7-1	1,416	1,317	-----	-----
7-2	2,567	2,633	-----	-----
7-3	9,016	8,867	-----	-----
7-4	6,650	7,747	-----	-----
7-5	12,333	14,966	1,319,500	-----
7-6	1,800	1,934	-----	-----
8-1	1,383	1,516	-----	-----
8-2	1,717	1,683	-----	-----
8-3	2,683	2,783	-----	-----
8-4	29,767	29,250	-----	-----
8-5	37,750	38,850	-----	-----
9-1	2,117	2,066	-----	-----
9-2	2,250	2,200	-----	-----
9-3	3,700	3,200	-----	-----
9-4	5,950	5,417	-----	-----
9-5	4,350	9,200	-----	-----
9-6	558,317	900,983	-----	-----

Table DBM.11 - Response Time - Multiple Relation Queries
Level 1 Indexes

Time to Last Record in Milliseconds				
Query	Database Size			
	3.5MB	6MB	10MB	56MB
6-1	3,317	4,867	687,450	-----
6-2	2,317	3,934	186,450	-----
6-3	2,167	3,283	357,550	-----
6-4	2,666	3,900	822,867	-----
7-1	169,283	319,383	-----	-----
7-2	26,800	57,950	-----	-----
7-3	180,483	344,083	-----	-----
7-4	94,100	189,850	-----	-----
7-5	57,650	99,983	-----	-----
7-6	217,650	411,367	-----	-----
8-1	510,300	956,433	-----	-----
8-2	25,650	36,116	-----	-----
8-3	178,400	341,900	-----	-----
8-4	117,500	217,033	-----	-----
8-5	345,717	663,750	-----	-----
9-1	528,350	851,616	-----	-----
9-2	212,400	384,683	-----	-----
9-3	545,017	816,567	-----	-----
9-4	12,916	20,267	-----	-----
9-5	4,367	9,216	-----	-----
9-6	558,434	901,000	-----	-----

Table DBM.12 - Response Time - Multiple Relation Queries
No Indexes

Response Time in Milliseconds		
Query	Time to First	Time to Last
	6MB	6MB
6-1	109,650	617,567
6-2	107,817	107,834
6-3	199,050	199,066
6-4	145,100	693,250

Table DBM.13 - Response Time - Update Queries

Query	Response Time in Milliseconds				
	Database Size and Index Level				
	6MB Level 1	10MB No Indexes	56MB No Indexes	56MB Level 1	56MB Level 2
I-1	1,150	850	1,116	900	1,400
I-2	1,333	983	1,266	1,500	1,886
I-3	1,333	950	1,216	1,333	1,366
D-1	13,550	13,830	120,383	118,916	119,683
D-2	866	800	188,766	900	1,116
D-3	1,166	11,650	72,200	70,266	72,116
M-1	933	916	-----	1,000	1,200
M-2	966	11,766	73,500	20,216	71,850

APPENDIX D - BENCHMARK SYSTEMS

D.1. MICROCOMPUTER DATABASE SYSTEM

D.1.1 SELECTION

Few database systems exist which are truly relational and run on microcomputers such as the PDP 11/23 computer that was used in the project. Many systems such as dBASE II and CONDOR are intended for use in small personal computers and have limited function and capacity. A sample of more qualified systems includes: MRS, MARATHON, and SEQUITUR. In this study MRS was chosen as the representative of the microcomputer architecture for the following reasons:

1. MRS is a fully relational database system as judged by the criteria published in [CODD 82].

2. MRS uses a subset of SQL as its query language. SQL [DATE 81] is widely used in larger relational database systems.

3. MRS is a stable system widely used in a university environment. MRS was developed by a highly regarded University of Toronto group and runs under the mini-UNIX operating system from Toronto.

4. The principal investigators had nearly one year of experience in using a version of MRS at the Database System Research Laboratory of the University of Maryland.

D.1.2 SYSTEM CONFIGURATION

An MRS database system running the Bell UNIX version 6 on a PDP 11/23 was benchmarked to represent the performance of a database system in the microcomputer environment. The size of the main memory on the PDP 11/23 was 256 KB. 10 MB of disk storage was used by the system. The database was stored on a dedicated disk of 10 MB.

D.1.3 IMPLEMENTATION AND BENCHMARK EXECUTION

The MRS system was developed at Computer Systems Research Group of the University of Toronto. Users, from a UNIX terminal, can access MRS by using English-like commands to query the database as well as to update it. It was initially designed for 'small' computers, where the maximum number of records in a relation is 30,000. Thus, the test database size was limited to 3.5 MB. Preliminary data preparation of transferring the benchmark database into the MRS format was required before any queries could be run using MRS. It should have been routine, but all the data and delimiters provided were not in a regular format. The first three weeks were spent transferring data and debugging. The next five to six weeks were spent writing and testing the queries.

MRS uses an SQL-like query language but it supports only an SQL subset. Thus, several variables considered in the other two benchmark studies were not included in the microcomputer test. These included sorting on primary keys, clustered indexing, more than two relation joins, concurrent multiple users, and background workload.

When interpreting the results of the microcomputer database system benchmark, several points should be recognized:

1. Limited Work Space - The original design of the work space (less than 10K) was for temporary storage of data from secondary storage before it was sent to the user. The space was also used by pointers retrieved from indexes. There was no problem when the queries did not use any indexes. But as additional indexes were added, the work space was quickly filled by index pointers, the query was aborted, and a warning message displayed.

2. Multiple Relation Join - MRS does not allow joins between more than two relations. Queries of this type were modified to queries in nested form. The records retrieved from the 'inner' query had to be stored in the work space used in the 'outer' query. Again, the work space was quickly filled, giving a warning message. To implement this test, the number of records retrieved was reduced.

3. Single User Restriction - The number of users on the UNIX operating system was restricted to a single MRS user. Although UNIX can support multiple users, this restriction was employed to simulate more exactly a true microcomputer architecture. Also, MRS does not provide concurrency controls for two or more users. The total elapsed time reflected the running time on a dedicated system for each specific query.

4. Query Access - MRS provides only two access methods, sequential and index access.

D.2. MINICOMPUTER DATABASE SYSTEM

D.2.1 SELECTION

There existed a greater number of choices for minicomputer-based relational database systems. A sample of these included: ORACLE, INGRES, ENCOMPASS, and QBE. Among the systems that run on the VAX 11/750 computer, ORACLE and INGRES are the best known. The original INGRES was developed by a research project in Berkeley. That system runs under UNIX but has many shortcomings [STON 80]. Since its performance did not represent a ´typical´ relational database in this class, it was eliminated from further consideration. Only the commercial version of INGRES, which is an enhanced version of the original INGRES, was considered. Both INGRES and ORACLE presently run under the UNIX operating system. They are in many respects very similar. In this study ORACLE was selected to be the target system for the minicomputer architecture for the following reasons:

1. ORACLE has been available commercially for several years and appears to have a larger user population.

2. Like MRS, ORACLE uses the SQL query language. This simplified the design of test transactions and job scripts.

3. The principal investigators had extensive experience in using ORACLE.

D.2.2 SYSTEM CONFIGURATION

An ORACLE Database System [ORAC 83], version 3.1.1, was installed on a VAX 11/750 running VMS 3.0. The 11/750 contained 2 MB of main memory. ORACLE memory requirements were 300 KB above the memory required for VMS (ORACLE VMS Installation Guide, p.1). The default sizes for all of ORACLE's work files were used except for the before images file (ORACLE$BI) which was increased to be equal to the largest database size to be tested (10 MB).

The mass storage available on the VAX 11/750 consisted of a Digital Equipment RL02 system disk, and a 9766 Control Data Corp. disk drive with an unformated capacity of 300 megabytes.

D.2.3 IMPLEMENTATION AND BENCHMARK EXECUTION

Version 3.1.0 of the ORACLE Database System [ORAC 83] was installed on a VAX 11/750 at the National Bureau of Standards during the second week of September 1983. Following the installation, initial attempts to create a 3.5 MB database failed due to a lack of space within ORACLE. This problem was resolved by creating a database partition large enough to handle the test database sizes (3.5, 6, and 10 MB) and adding it to ORACLE.

The base relation table was then loaded with data for a 3.5 MB database (10,500 records) using procedures developed by Systems Development Corp. (SDC). Attempts to load the individual database relations failed at this point. The database loading procedure required that several update operations be performed on the single large relation prior to loading individual relations, and ORACLE locked while performing these updates. ORACLE technical services personnel suggested that the problem might involve one of ORACLE's work files, the before images file, which could be too small to handle updates on large relations. The size of the before images file was increased from 1 to 4 MB and the update operations were successfully run.

The 3.5 MB database was then used to test and debug the query sets. This took six weeks, since the queries were not returning results identical to those returned by the database machine. The reasons for these discrepancies turned out to be several minor logic problems with the database load procedures developed by SDC. These problems were corrected and testing began the first week of November.

Running queries against the 3.5 MB database proceeded smoothly until joins were attempted on tables which had indexes formed with concatenated fields. Query set 8 (joins) ran for more than 10 hours without producing results. Discussions of the problem with ORACLE technical services people revealed that version 3.1.0 of ORACLE had a known bug with concatenated index fields, which had been corrected under version 3.1.1. It was decided to install version 3.1.1 and rerun the queries to allow the ORACLE database to have indexes identical to those used by the database machine. After installing 3.1.1 and rerunning the queries, though, there was still a problem with concatenated index fields. As a result every index with a concatenated field (social security number plus an additional field) on the database machine was created with only one field under ORACLE. The joins were then rerun and all but query set 10 (4 relation join) completed successfully.

After testing on the 3.5 MB database was complete a 6 MB database with 22,000 records was created. All the single relation queries and all multiple relation queries (except joins without indexes and query set 10) were run and completed. Additional tests performed on the 6 MB database included multiple user performance and performance as affected by background jobs. Except for queries run in multi-user mode or explicitly run with background jobs, there was no background load on the VAX at any time (except a minimal amount of time required by the operating system to maintain system processes). Several joins without indexes were allowed to run to completion. These joins took 10 to 12 hours to run against the 6 MB database; joins with level 1 and level 2 indexes finished significantly faster.

Testing queries against the 10 MB database posed no problems except that all queries run without indexes did not finish within the 30 minute time limit. The size of the before images file also had to be increased from 4 to 11 MB to accomodate the updates run against the 10 MB base relation during the database creation process.

Most problems encountered with ORACLE involved default space allocations, which were typically too small to handle databases of any real size. The only problem for which ORACLE did not issue an error message involved running out of room in the before images file. Problems with indexes were not apparent until queries actually failed to run, though, since ORACLE does not automatically validate them when they are created (a specific request by the user is necessary to validate an index once it is created).

D.3. DATABASE MACHINE

D.3.1 SELECTION

There are few database machines available commercially. The best-known machines are Britton-Lee's IDM and Intel's iDBP. The IDM is a relational database machine using a QUEL-like machine language. The iDBP supports lower level operations and also handles variable length records and string searches. For this performance study the IDM-500 database machine was selected for the following reasons:

1. IDM has been available for several years and at the present time over 100 units have been delivered and installed. On the other hand the iDBP was only recently announced (in 1982), and has no commercially-available front-end (host) software, and is not widely used.

2. IDM is a relational database machine. This simplified the benchmark design and result interpretation and facilitated the comparison with other relational database systems.

3. There existed a variety of host software for the IDM. For this study, the host software provided by the vendor was used.

4. The principal investigators had used the IDM-500 system. Host software and an SQL processor for the IDM was developed as part of a research project at the Database Systems Research Laboratory of the University of Maryland.

Britton-Lee manufactures two models of the IDM, the 500 and the (newer) 200. Both are identical from a functional point of view. The IDM-500 version was selected because of its larger capacity and better availability based upon time constraints. Britton-Lee has developed an add-on to the IDM-500 which they call a "hardware accelerator". It is expected to greatly increase the speed of the IDM-500. However, for this study the IDM-500 without the accelerator was benchmarked.

D.3.2 SYSTEM CONFIGURATION

An IDM-500 database machine (110 volt model) with one MB of memory was installed and used for the benchmark. After loading the IDM-500 software, approximately 1/2 MB of memory was available for users. Release 24 of the IDM software was used. Mass storage for the IDM consisted of a 9766 Control Data Corporation disk drive with an unformated capacity of 300 MB. The disk drive was ported to the IDM-500 via an SMD interface with a data transfer rate of 1.2 MB per second.

The IDM operated with a VAX 11/750 "front-end" computer running Berkeley UNIX 4.1. The data transfer rate between te VAX and the IDM-500 was through a 9600 baud RS232 interface.

D.3.3 IMPLEMENTATION AND BENCHMARK EXECUTION

The IDM-500 database machine was delivered to NBS during the third week of May, 1983. Britton-Lee arrived the next week to install the disk drive and load the software.

Initially, a 10 MB Control Data Corporation disk drive, SMD compatible, was to be used as the IDM drive, but all efforts to hook the drive into the IDM failed. Eventually, it was decided to use a Control Data Corporation 300 MB drive. The drive was cabled, initialized, and the software unloaded.

Sometime that night a power surge affected the IDM and/or the disk drive, and the disk format was lost. Britton-Lee returned two days later and reinitialized the disk. This problem fortunately never reoccurred.

The IDM single user software, IDL, was also installed at this time. IDL and the query script runner program, ´runner´, (written by SDC) were tested. Runner was locally modified to reflect device names and timing algorithms.

The data tape was prepared by SDC and the entire tape, 189,960 records, was loaded into the IDM-500. This initial load took over 12 hours. Breaking up the relations and adding indexes took another 16 hours.

The next 4-6 weeks was spent writing and testing queries. By July 1, the final benchmark workload was written and testing began. No problems occurred during testing of the single relation queries, but it quickly became apparent that the multiple relation queries (the joins) exhibited widely different performance depending upon the database size.

After testing on the 56 MB database was complete, the benchmark tests were duplicated on different size databases. The single base relation ´nbsol´ was copied onto the UNIX disk, via the IDM Fcopy command, and a new database, 3.5 MB, consisting of the first 10,500 records of the file was created. It took approximately four hours to build this new database, and testing began on August 10th. It was apparent almost immediately that the queries were being processed much more quickly on the smaller database.

Query set 10, the four relation joins, posed a problem. None of the queries completed within the 30 minute time frame, and when tested under IDL, these queries ran very strangely. They would start to print normally, but would stop in the middle of a record.

A 10 MB database containing 33,000 records was created. This database handled much the same as the 56 MB database - the joins took an extremely long time to run. All single relation queries ran fine. It appeared that somewhere between 3.5 and 10 MB the IDM-500 could no longer efficiently handle joins between large relations. In order to determine where the break point occurred, a 6 MB database (20,000 records), a 7 MB database (22,250 records), and a 8 MB database (25,500 records) were benchmarked. The performance break point in running the joins appeared to be between the 7 MB database and the 8 MB database.

Two additional 9600 baud lines and ports were then dedicated to the IDM to test muli-user functions. During testing (via the ´runner´ program), the order of the queries in the job scripts had an effect on performance when three users were on the system. (This was discussed in Section 8.2.4.)

Some additional benchmark execution problems included:

1. The IDM-500 and the disk drive were extremely sensitive to electrical current fluctuation. Even if the IDM stayed up, which it usually did, contact with the disk was lost, and a reboot was necessary.

2. A non-super-user on the UNIX system could not KILL an IDL process. This hindered testing.

3. The RS232 communications were very slow between UNIX and the IDM. At 9600 baud (about 900 characters per second) it took 26 hours to load the 56 MB database. Database loads of the other database sizes were correspondingly slow.

☆U.S. GOVERNMENT PRINTING OFFICE: 1985-461-105/20247

NBS-114A (REV. 2-80)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. NBS/SP-500/132	2. Performing Organ. Report No.	3. Publication Date October 1985

4. TITLE AND SUBTITLE

Computer Science and Technology:
Benchmark Analysis of Database Architectures: A Case Study

5. AUTHOR(S)

S. Bing Yao and Alan R. Hevner, Software Systems Technology, Inc. & Daniel R. Benigni, Editor

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

~~NATIONAL BUREAU OF STANDARDS~~
~~DEPARTMENT OF COMMERCE~~
~~WASHINGTON, D.C. 20234~~

Software Systems Tech., Inc.
7100 Baltimore Avenue, Suite 206
College Park, MD 20740
(301) 779-6030 (or 5486)

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

National Bureau of Standards
Department of Commerce
Gaithersburg, MD 20899

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 85-600599

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

The purpose of this guideline is to present an application of the generalized performance analysis methodology for the benchmarking of database systems that was reported in NBS Special Publication 500-118. The principal objectives of this guide are to benchmark the performance of three distinct database system architectures: 1) a microcomputer database system; 2) a minicomputer database system; and 3) a database machine. This guide not only proves the viability of the benchmarking methodology in evaluating real systems, but it also provides comparable observations as to the capabilities of database systems based upon different architectures.

Together with NBS Special Publication 500-118, this report serves as a reference for the benchmarking of database systems by providing a complete description of the benchmarking framework and a detailed application showing how to implement it.

(Related Documents: NBS/SP-500/118: A Guide to Performance Evaluation of Database Systems by Daniel R. Benigni (Editor), S. Bing Yao and Alan R. Hevner; NBS-GCR-84-468: An Analysis of Three Database System Architectures Using Benchmarks by S. Bing Yao and Alan R. Hevner)

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

benchmark execution; benchmark methodology; benchmark workload; database systems; DBMS; indexing; performance evaluation; query complexity; response time

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

198

15. Price

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, DC 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name ______________________________

Company ______________________________

Address ______________________________

City ______________ State ______________ Zip Code ______________

(Notification key N-503)

NBS Technical Publications

Periodical

Journal of Research—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. Issued six times a year.

Nonperiodicals

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396). NOTE: The Journal of Physical and Chemical Reference Data (JPCRD) is published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements are available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service, Springfield, VA 22161, in paper copy or microfiche form.

U.S. Department of Commerce
National Bureau of Standards
Gaithersburg, MD 20899