NASA-TM-83942

NASA-TM-83942 19830014922

NASA

Technical Memorandum 83942

Data Base Management System Analysis and Performance Testing with Respect to NASA Requirements

E. A. Martin, R. V. Sylto, T. L. Gough, H. A. Huston and J. J. Morone

AUGUST 1981

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt, Maryland 20771

National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland
20771

NASA

Reply to Attn of: 931

FILE 14281

TO: Distribution

JUL 28 1982

FROM: Applications Directorate
Information Extraction Division

SUBJECT: Release of Report "Data Base Management System Analysis and Performance Testing with Respect to NASA Requirements"

The purpose of this memo is to announce the release of Technical Memorandum No. 83942, "Data Base Management System Analysis and Performance Testing with Respect to NASA Requirements". This report was jointly produced by Business and Technological Systems, Inc. under contract to GSFC's Information Management Branch, and by Ms. Elizabeth Martin and Ms. Regina Sylto from the Information Management Branch as part of the GSFC's NEEDS DBMS Technology Task.

This report is an evaluation of the use of several data base management systems in the context of NASA satellite data systems applications. In order to meet the requirements of the Information Extraction Division (IED), particularly the development of the NEEDS Packet Management System (PMS), the Information Management Branch conducted a study which compared two commercial and one NASA-developed data base management systems --ORACLE (Relational), SEED (CODSYL Network), and RIM (Relational-NASA LaRC). The results of this study are intended to aid IED personnel and other data system developers in selecting data base management systems (dbms) that will perform well for various NASA satellite data system applications.

It is important to note that the defined goals of the study were to assess the capability of the dbms to manage large amounts of data (i.e., at least a million input records), to determine ingestion rates, to measure the efficiency of the various data access techniques used in the dbms, and to evaluate qualitative characteristics of the systems considered. Because of schedule, available dollars, and a need to support the development of the PMS, strong emphasis was placed on determining what impact using large volume data bases has on the performance of the dbms. Since large amounts of actual NASA satellite data were limited, the study focused on a single data base application which managed stratospheric temperature profiles (LIMS data).

After the completion of this initial study, we have found that other types of data base applications may produce different performance results. The Information Management Branch has initiated a follow-on study under the DBMS Technology Task that will determine the effects of varying additional factors (e.g. VAX resource allocations for a dbms package, number of fields, record length, number of key fields) and what impact these factors have on dbms performance. The results of this new study are expected to be available for release in early 1983.

Copies of TM 83942 have only been included with this memo for those people who have requested the report. A limited number of additional copies are available. If anyone else would like a copy, please contact Regina Sylto, Code 931.2, telephone 301-344-9040.

Elizabeth A. Martin

Elizabeth A. Martin
Information Management Branch

CONCURRENCE

Paul H. Smith

Paul H. Smith, Head
Information Management Branch

DATA BASE MANAGEMENT SYSTEM ANALYSIS AND PERFORMANCE TESTING WITH RESPECT TO NASA REQUIREMENTS

PREPARED IN COOPERATION WITH THE
NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

FINAL REPORT
CONTRACT NAS 5-25561

BY
T. L. GOUGH
H. A. HUSTON
J. J. MORONE

BUSINESS AND TECHNOLOGICAL SYSTEMS, INC.
AEROSPACE BUILDING, SUITE 440
10210 GREENBELT ROAD
SEABROOK, MARYLAND 20706

AND BY

E. A. MARTIN
R. V. SYLTO

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND 20771

AUGUST 6, 1981

N83-23193#

FOREWORD

As part of the NASA End-to-End Data System (NEEDS) program to demonstrate a more efficient and timely transfer of data from sensor to user, the Information Extraction Division (IED), Goddard Space Flight Center, has been responsible for the technical direction associated with the development of the Packet Management System (PMS). The PMS is to be responsible for managing a catlog of packet headers and for interfacing with end users for browsing and retrieving data from an Archival Memory. This component is one of several in a system referred to as the NEEDS Phase II Data Base Management System. The PMS system was originally titled the Integrated Data Base Management System (IDBMS) and was renamed after an alteration in requirements.

To meet the needs of the IDBMS, IED personnel conducted a study of commerically available Data Base Management Systems (DBMS) that could operate on the Digital Equipment Corporation (DEC) VAX-11/780 computer. As a result of this study, two systems were procured, ORACLE and SEED, for consideration as a nucleus of the IDBMS. It was then that this study was originated to make a comparison of these systems' abilities to meet the needs of a NASA application such as the PMS or IDBMS. Subsequent to the study's inception a third system, RIM (part of a NASA computer-aided design R&D effort to develop technology for management of engineering information) was added for consideration. The results of this study are intended to support the decision of how best these systems can be applied to support NASA needs especially in relation to PMS.

The study has been conducted under the technical direction and review of Elizabeth A. Martin of the IED. The document has been prepared by Thomas L. Gough (Project Manager), Herbert A. Huston, and John Morone of Business and Technological Systems, Inc. (BTS), and by Elizabeth A. Martin and Regina Sylto of the IED. Mary Reph (IED) also has contributed significantly to the study effort and the assisitance of Paul A. Maresca (BTS), and Karen Posey (IED) is greatly appreciated.

Prior to this document's distribution, it was submitted for review to the originators of each of the DBMS's. Their comments have been included as APPENDIX III to permit a rebuttal to any aspects of the study.

TABLE OF CONTENTS

TABLE OF CONTENTS (CONTINUED)

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This study was conducted to evaluate several candidate Data Base Management Systems (DBMS's) that could support the NASA End-to-End Data System's Integrated Data Base Management System (IDBMS) Project which was later rescoped and renamed the Packet Management System (PMS). The candidate DBMS systems which had to run on the Digital Equipment Corporation VAX 11/780 computer system were ORACLE, SEED and RIM. ORACLE and RIM are both based on the relational data base model while SEED employs a CODASYL network approach.

Various constraints required the study to focus on a single data base application which managed stratospheric temperature profiles. The primary reasons for using this application were an insufficient volume of available PMS-like data, a mandate to use actual rather than simulated data, and the abundance of available temperature profile data. Goals of the study included the following: to assess if the systems could manage large amounts of data (i.e. at least a million input records), to determine ingestion rates, to measure the efficiency of the various data access techniques used in the systems and to evaluate qualitative characteristics of the systems considered.

The test plan employed called for five staged loads for each of the three DBMS's under study. After each staged load a set of tests were repeated that exercised certain capabilities of each system. The number of input records managed by the DBMS's grew from about 50,000 after the first staged load to over a million after the fifth. The tests were conducted in a "stand-alone" mode to eliminate uncontrolled biases. One should realize that results obtained in this mode are presumably "best case" numbers.

Generally, the load results indicate that SEED is significantly faster than RIM which is similarly faster than ORACLE below the half million record level for this application. From a half million to a

million records the incremental load rates begin to favor ORACLE and at over one million records, ORACLE is clearly superior. (This is paradoxical to conventional theory regarding relational and network data bases.) The reader must penalize the RIM load results because the RIM data base design had to be amended to delete one indexed character field because of the inefficiency in managing duplicate or psuedo-duplicate key values. Of interest and possible concern is the amount of observed CPU utilization by each of the systems. ORACLE, SEED and RIM consumed approximately 75%, 50% and 34% of each available CPU second, respectively, implying ORACLE would be more seriously impacted by other users than SEED or RIM.

Each system varied significantly in the amount of space required to manage the one million records. ORACLE required 87 million bytes, partially due to a module 64 byte memory management scheme (promised by the vendor to be improved in a later software release), SEED required 20 million bytes and RIM used 40 million bytes.

In general, the ORACLE and RIM systems are much simpler to grasp and use. There is less emphasis placed on data base theory and more flexibility in making changes to a data base design without paying heavy restart prices. SEED on the other hand requires much greater comprehension of data base concepts by the data base designer or user. It does offer a greater degree of customization for a particular data base application which may lead to higher performance but typically is not easy to modify if a design change is needed after implementation. RIM's current lack of support for multiple users may well eliminate it from consideration in many applications and its lack of complimentary software (i.e. report writer, data entry, etc.) also weighs against it.

The study's results indicate that none of the candidate systems fully meet the original requirements of the IDBMS. But, clearly, no commercially produced and marketed DBMS system now available on any computer system could support all the unique needs of the IDBMS. The results do indicate that these systems could be used as a central core to an IDBMS-like system around which additional software would have to be built to satisfy many of the specialized requirements of the application. Readers should be warned that the results from the tests are based on a single application and should be aware that other applications may produce significantly different numbers.

1.0 EXECUTIVE SUMMARY

The function of this preliminary section is to summarize the overall DBMS benchmark effort and results in a relatively high level manner for those members of the reading audience who wish to get a brief description of the study. For those interested in the details of the study, section 2 describes the testbed, test procedures, and test results. Section 3 discusses the DBMS systems in terms of their ability to meet certain qualitative requirements thought desireable in a NASA data base application and Section 4 offers a more detailed summary than this section.

The purpose of the study, simply stated, was to evaluate the performance and characteristics of several Data Base Management Systems (DBMS) software packages that were available and could execute on the VAX 11/780 computer. The study was conducted for the Information Extraction Division (IED) of the Goddard Space Flight Center and was specifically concerned with the systems' abilities to meet requirements of the Integrated Data Base Management System (IDBMS) which was part of the NASA End-to-End Data System (NEEDS) project. These requirements were defined in the "NEEDS Data Base Management System Functional Requirements" dated March 20, 1980, and Appendix IV contains a reprint of chapter three which specifies the requirements. Before the completion of the DBMS study the IDBMS had been redefined and renamed the Packet Management System (PMS) and new requirements for the PMS were not available in time to be reflected in this study specifically. However, the results of this study are still relevant for aiding in making decisions associated with the DBMS aspects of the PMS.

A DBMS might be simply described as a system which attempts to collect and organize information which can be later identified or located for reference, update, or deletion. The state of the art in software technology has progressed to where a number of approaches to data base models have evolved and numerous software systems with varying degrees of sophistication have been implemented using the different approaches.

Initially, the IED chose two commercial systems for the DBMS study based on an investigation of available DBMS systems that could operate on the VAX 11/780 computer. The two systems chosen were ORACLE, marketed by Relational Software, Inc. of Menlo Park, Ca., and SEED, marketed by International Data Base Systems, Inc. of Philadelphia, Pa. The two systems were designed from different data base models that are highly divergent in the manner in which data is organized. ORACLE is based upon the relational data base model which presents information in the form of two dimensional tables. SEED is based on the CODASYL network specification and presents information as sets that have members and owners. The two systems were both selected because of the sophisticated capabilities possessed by both including indexing techniques to accelerate data location, interactive query language for online data access, and various utility routines for added DBMS power to name a few.

A late entry to the study was RIM, developed at the Langley Research Center as part of a NASA supported joint industry/government project denoted Integrated Programs for Aerospace-Vehicle Design (IPAD). The development of RIM is part of a computer-aided design (CAD) research and development effort to develop technology for management of engineering information. RIM is based on the relational data base model and, as such, shares many attributes in common with ORACLE but, at least in the initial delivery, was far less sophisticated than either ORACLE or SEED. This might be expected if one assumes that at least initially it was developed for a primary application and that it was not intended to be a commercially marketable product for general use.

Several problems were present during the testing which affected the study. A major problem was a lack of resources (time, computer, and manpower) available to support the design and performance of an exhaustive set of tests to evaluate DBMS capabilities. Further, both ORACLE and SEED were repeatedly updated with new versions resulting in test delays and thus further reducing available time. IED personnel requested that the testing be based on actual NASA data but the more appropriate data sources (e.g., PMS-type data, other Applications' data catalogs and inventories) did not exist in substantial enough amounts for testing. The choice was then made to use satellite sensor data. The final selection made was to use FGGE/LIMS data containing stratospheric temperature profiles from the NIMBUS 7 mission. There were two primary types of records in this data, a Profile record and an Entry record. A Profile record contained a time, latitude, and longitude for a specific temperature profile and was associated with a specific magnetic tape. Approximately 16 Entry records were present for each Profile and, each containing a pressure level, pressure type, temperature, and quality control indicator for a specific level within the profile. When reference is made in this report to "number of records in the data base" it is in the context of the above description. By describing the number of records by reference to the input data format instead of actual records in the data base, comparisons between data bases can be made more easily. This approach transcends, for discussion purposes, unique DBMS implementations that might require multiple data base records per single input record or vice versa. It is considered appropriate to do this since the data being managed is the same for each DBMS in the study.

After the choice of the data base application was made a set of logically similar designs were derived for each DBMS, implying that fields selected for indexing in one were indexed in all DBMS's and that data relationships were maintained synonomously. The simplicity of the FGGE/LIMS data enabled the creation of data base designs that provided a suitable testbed which permitted comparisons between the systems' performance. It must be stated, however, that the use of a single application for total system evaluation is unwise. As stated later in this section in more detail, an expansion of testing is required to fully explore total system performance.

The test plan that was applied for this study recognized that resources did not permit a comprehensive testing of all facets of DBMS capabilities. The test plan concentrated on accomplishing several goals. One goal was to determine how sensitive the systems were to size. This was deemed important because of the typically large amounts of data associated with many NASA applications including the PMS. A second goal was to compare the loading performance of each system because of the high data rates associated with the PMS. A third goal was to determine the efficiency of the data accessing techniques implemented in the DBMS's. Emphasis was placed on access to data versus update or deletion which is consistent with many large scale NASA data base applications as well. Another goal was to evaluate the systems in a qualitative sense to identify characteristics such as flexibility, user friendliness, control and complimentary functions.

It should be noted that the applications foreseen for the DBMS system at NASA are atypical. The scientific environment and the high volume of data that is normally static once incorporated in a data base results in different emphasis than perhaps a business oriented application such as a corporate information management system or reservation system. The application dependence of a system's performance must not be underestimated and guarded conclusions should be made when looking at test results gained from an application foreign to one's own.

Another goal of the tests was to exercise both the terminal interface capabilities as well as high level computer language interfaces. Terminal Interface (TI) capabilities existed in all three systems and the interactive features are very important for effective use of the data base by users. The high level language interface or host language interface (HLI) is also important because the highly specialized needs of the PMS require customized software that can communicate directly with the DBMS selected for incorporation with it.

The final test plan met these goals by proposing a relatively concise set of functions including queries, deletes, and insertions using both the TI and HLI capabilities for each system. In addition these functions were to be performed with differing amounts of data in the data bases. Thus, each DBMS was used to load a prescribed amount of FGGE/LIMS data, was subjected to parallel sets of tests for the TI and HLI, and then was loaded with more FGGE/LIMS data to have the test repeated again. Due in part to the amounts of data present on the FGGE/LIMS data tapes the following numbers of records were used to repeat the tests: 52,000, 99,000, 189,000, 439,000 and 1,040,000. Since there were three data bases with two sets of tests for each (TI and HLI) at each data base level and there were five levels of data base size, 30 sets of tests had to be performed.

To make the results meaningful a controlled environment was required to eliminate unknown variables. Since there was no way to objectively factor the impact of other VAX computer users or to control their activity so test results would be repeatable, all tests were conducted during the third shift or on weekends when the computer could be reserved for data base testing only. The results obtained should be repeatable under these circumstances and introduce no bias that would have positively or negatively influenced one system verses another. The results should be "best case" results for the application since there were no other users contending for system resources.

The load rates of the three systems varied somewhat over the course of the one million records loaded into each data base. The ORACLE system varied the least over the loading processes. A slight degradation is detectable but the initial rate of 7.4 records/sec only dropped by about 1.2 to 6.2 records/sec at the largest data base size. The SEED load rates began at a much higher level (45 records per second initially) but showed a great deal of fluctuation. A substantial degradation occurred during the course of the loads. Around the 1 million record level the rate fell below the 5 record/sec level. The RIM system design had to be altered so that the Profile records time field was not indexed after the initial 52,000 record load because of an overhead associated with indexing data with characteristics common to the time values. At the 99,000 record level the profile time index was removed from the data base the costs at the low end are reasonable, or that for relatively small data bases SEED may have an advantage. SEED's degradation is attributed, in part, to a default algorithm used to determine a location in the data base for indexed values to reside. Improvement in performance may be obtainable by a user defined algorithm (permissible in SEED) that improves the location process by more uniformly distributing the data locations because of prior knowledge of the characteristics of the data. Also the SEED system has a variety of options selectable in the data base design which can impact the load rates, as well as query responses. Several of these options are discussed briefly in Section 2 but tests have not been conducted to formally compare each option. The options available have the advantage of offering different approaches for data bases with differing characteristics and needs. However, a naive or unfamiliar individual may use options that reduce performance if he is not careful. Neither ORACLE nor RIM offer options for how or where the data can be stored.

Noteworthy is the fact that during the loading which was almost always done in the absence of other VAX users (eliminating contention) each system utilized the CPU to different degrees. The ORACLE loads used about 75% of each second of available CPU time. SEED used slightly less than half of each second on average and RIM used as much as 34% during early loading but dropped to about 24% during the final load. The implication here is that if users were contending equally for CPU time then ORACLE would be impacted more heavily than SEED, and RIM would be less impacted than SEED. Obviously all systems would perform at lower levels of efficiency when sharing resources with other users.

The TI query results should be considered in the context of their use. An interactive user is not concerned with extremely short delays in response. A pause of a few seconds is not unacceptable in most interactive situations. The results of tests indicate that the indexing techniques implemented by all three systems are adequate in providing responses that are acceptable for interactive users at all levels of data than half of each second on average and RIM used as much as 34% during early loading but dropped to about 24% during the final load. The implication here is that if users were contending equally for CPU time then ORACLE would be impacted more heavily than SEED, and RIM would be less impacted than SEED. Obviously all systems would perform at lower levels of efficiency when sharing resources with other users.

The TI query results should be considered in the context of their use. An interactive user is not concerned with extremely short delays in response. A pause of a few seconds is not unacceptable in most interactive situations. The results of tests indicate that the indexing techniques implemented by all three systems are adequate in providing responses that are acceptable for interactive users at all levels of data base size. The results also indicate that unacceptable delays are encountered when queries are made that must search through non-indexed values. A good analysis and data base design effort must attempt to eliminate the need for queries that require searching of large groups of records by properly specifying indexed fields. The results do show that RIM is decisively faster than either other system in performing a search through all ocurrences of a particular non-indexed data field. They also show that SEED is consistently faster than ORACLE.

The HLI results for the queries can be examined more closely since in this mode relatively small differences can result in large cumulative differences when a piece of software is repetitively performing functions that require an interface with the data base. The results generally indicate that SEED can locate indexed values consistently faster than RIM which can locate indexed values faster than ORACLE. The magnitude of the SEED responses for locating a particular profile time for cases where there are from 3000 to 26000 possible values is around .1 seconds while with about 60,000 possible values it was about .5 seconds. For RIM it is around .3 seconds until the 60,000 level when it increments to about .5 seconds also. The ORACLE results are around .5 seconds for all levels. While trying to locate a particular Entry record SEED requires around .3 seconds when selecting from among 49,000, 93,000, 178,000 and 413,000

Entry records. With about 980,000 choices to contend with, it located the desired record in about .75 seconds. RIM required about .5 seconds at the lower levels and about .75 seconds also at the 980,000 level. ORACLE required 1.08 and 1.33 seconds for all levels. The differences may be small by themselves but if software interfaced with the data base and another computer and had to handle bursts of requests the cumulative difference could become significant.

HLI tests which accessed all occurrences of a particular data item might be equated to either the querying of non-indexed fields or the production of summaries about a field or record. The results indicate that RIM is substantially faster than either ORACLE or SEED. To access all 980,000 Entry records when the data base was at its largest ORACLE required over 3 hours and 42 minutes, SEED required over 1 hour and 3 minutes and RIM took less than 17 minutes. If periodic reports or summaries are frequently generated for an application, these results may be worth considering when making a DBMS selection. For example, if daily summaries were produced which required the processing described above almost a sixth of available processing time would be spent with ORACLE while less than two percent would be spent with RIM (in an uncontested environment). The matter of storage utilization was also addressed and it was found that ORACLE consumed about 87 million bytes to manage the 1,040,000 Profile and Entry records in the largest data base (60,000 Profile records and 980,000 Entry records). Of the 87 million almost 43 million is wasted due to the current storage management capability in ORACLE. This is said to be corrected in the 3.0 version of ORACLE to be released in late 1981. SEED consumed under 20 million bytes to manage the same data and RIM consumed about 40 million bytes.

Some general comments regarding the systems are also worth stating. The ORACLE system was found to be a much simpler one to grasp and use. A relative novice to data base theory could devise a feasible design and implement it using ORACLE for many applications with a small amount of training and/or research. The likelihood of serious flaws are small and the flexibility of ORACLE allows for design modifications without requiring starting over from "square one." SEED is generally the opposite of ORACLE in this regard. A much deeper understanding of the CODASYL network specification and SEED capabilities and options are required for an individual to design and implement a data base. Any modifications to the design almost invariably require starting over with a specification of a new schema and the steps that follow that. RIM is similar to ORACLE in terms of comprehension of the data model but is not as complete or as thoroughly implemented as ORACLE.

Although RIM has shown certain capabilities that demonstrate great potential it does lack, to some degree, the generality of the other systems as well as the complimentary software available in the other DBMS's. A major shortcoming of RIM and one that should eliminate it from use in the PMS is that it does not support multiple users. It is unknown at this time if or how RIM will be modified in the future for support of its current or other applications.

The results of this study are somewhat inconclusive. The original goals have been met but the results have not supported the elimination of any of the DBMS's (except RIM for the reason stated above). It appears that none of the systems meet all of the original IDBMS requirements, but, in truth no general purpose DBMS has been produced that could support the unique needs of such a system. The study has provided a foundation from which some comparisons can be made and which may be used to better envision how the systems can be applied to support other needs. An overall benefit of this effort has been a greatly increased knowledge of how to apply the DBMS's as well as more awareness about DBMS usage and capabilities in the atmosphere of NASA applications. The results stated give some indications of upper limits of performance to aid in estimation of maximum throughputs possible with the data bases. The results also indicate that the systems are capable of managing relatively large amounts of data although performance is not consistent over all ranges of data base size.

Additional factors need to be examined in future studies to more thoroughly understand and predict the performance of the DBMS's including:

- Record sizes
- Number of indexed fields
- Size of an indexed field
- Experimenting with SEED options
- Experimenting with ORACLE data base parameters
- Multiple data base users
- Multiple VAX users

2.0 QUANTITATIVE ANALYSIS

The development of the test plan for benchmarking DBMS performance was constrained by several factors including:

- Computer resources (time and storage)
- The number of systems under scrutiny
- Calendar time available before results were needed
- The environment of the target application (IDBMS)

The number of systems under consideration and the impact on the Host VAX computer, in both consumption of time and of auxiliary memory, required careful definition of test procedures. Further complicating the matter was the urgency associated with the completion of the testing. The test plans had to be relatively concise because each DBMS would be exposed to them independently and repetitively.

The tests were performed on SEED, ORACLE and RIM data bases with fixed amounts of FGGE/LIMS satellite data. Originally the goal was to perform tests with approximately 50,000, 100,000, 200,000, 400,000, 1,000,000, and 2,000,000 million records in the data base. Ultimately the 2,000,000 goal was discarded and five test points were used: 52,0000, 99,000, 189,000, 439,000, and 1,039,000. By performing the tests repetitively over these ranges any sensitivities to data base size would become apparent. Since the target data base application would manage large amounts of data, the 2 million record level would have been desirable but the amount of disk space and the lack of time available made it impractical to proceed past 1 million.

2.1 Background and Environment

2.1.1 Data Base Application

To facilitate the quantitative measurement of DBMS performance, a single application was selected from which all results could be obtained. Although the data base packages were expected to be used to manage catalog data the FGGE/LIMS satellite data was selected because substantial amounts of catalog data were not available. The creation of artificial catalog data was discarded because it was considered undesirable to use test data when ample amounts of actual satellite data existed. A number of FGGE/LIMS tapes were available and offered enough data to demonstrate the management of large amounts of information. It was also felt that the use of satellite data would demonstrate PMS capabilities better than a more abstract application. The FGGE/LIMS data was stored on a number of magnetic tapes with each tape containing observations for an exclusive period of time. The tapes available included observations recorded from December, 1978, through May, 1979. A tape consisted of files, each of which corresponded to a particular six-hour time period referred to as a synoptic time period. Within the file were profiles identifying the time, latitude, and longitude associated with a set of observations which describe a vertical column of the atmosphere. The set of observations for a profile consisted of approximately 16 records referred to as Entry Records in this document. Each entry record consisted of a pressure type, pressure level, temperature, and quality flag.

To provide a fair basis for comparison, the data bases were all designed as similarly as possible. The relational systems, ORACLE and RIM, possessed the same basic capabilities and a similar design for each was a simple matter. Production of an equivalent design for the CODASYL system, SEED, was a more difficult task. Each of the systems under study offered a caspability to locate the occurrence of specified values for some data items without the need to sequentially search all the occurrences of that data item. Many methodologies have been developed to provide this function and many terms have evolved to describe them including data base keys, indexes, and images to name a few. For the sake of discussion, the capability shall be called direct access and data items which are specified to have this characteristic are referred to as indexed fields. To maintain consistency between the three designs any data item that required direct access was given that characteristic in each system. To accomplish this ORACLE and RIM use a technique called a

B-tree which refers to the data item values as keys. When a key value is referenced a binary search is made through a logical tree hierarchy of key values to locate the occurence of the desired value and, if it is found, a pointer(s) will be present locating the tuple(s) or row(s) which contain the value. In SEED direct access is accomplished through the use of a hashing algorithm technique. This approach performs an operation on the key value producing a numerical result which is a logical pointer to a location where the data record should reside in the data base. In SEED the smallest entity which can be directly accessed is a record. In the chosen application the profile record presented as input to the data base had time, latitude, and longitude values all of which required direct access. The SEED data base design had to include a record definition for each of these items to accomplish the direct access. This means that multiple record occurrences exist in the SEED data base for each Profile record input to it. For clarity the results in this document will refer to records in the data bases in terms of the way they were presented as input to the load software not in terms of the internal management used by each data base unless otherwise stated. The relative simplicity of the FGGE/LIMS data, i.e. the small number of data items and the clarity of the relationships between them, made the design of consistent approaches possible, thus the benchmark results do provide a basis for comparison of DBMS capabilities.

2.1.2 Data Base Design for FGGE/LIMS Data

2.1.2.1 ORACLE Data Base for Satellite Data

2.1.2.1.1 ORACLE Data Base Construction

After the decision was made to use FGGE/LIMS data for the DBMS benchmark, a design phase was conducted which considered factors including: the LIMS data; the volume of this data; the way this data could be accessed in the existing Climate Data Access System (CDAS) which already manages FGGE/LIMS data tapes for climate research; the need for sequenttial retrieval, and the facilitation of a test bed for benchmark analysis. Because of the small number of relationships that existed in the LIMS data, a complex system of tables is not required. Three sets of tables (Tables 2-1, 2-2 and 2-3) which are all somewhat similar were considered for the data base design.

All of the proposed sets of tables contain three common tables, TAPE, PRESSURE_TYPE_LEGEND, and QUALITY_FLAG_LEGEND. The TAPE table contains a row for each LIMS tape in the data base and has domains for: tape ID, start synoptic time, stop synoptic time and date of tape creation. The two legend tables give meaning of various values which the PRESSURE_TYPE and QC_FLAG fields have in other tables in the data base. These tables represent an initial attempt to normalize the tape data.

In addition to the three tables described above, the first proposed set of tables included a fourth table that contains the remainder of the data base information. The fourth table in the first set of tables is LIMSDATA and would contain about two million rows of data, if all the available data was loaded into it. (The two million figure was used because IED personnel desired to see the data base system perform with as much data as possible and this was approximately all the data on the available FGGE/LIMS tapes.) Each row of this table would contain the fields: tape ID, file synoptic time, profile time, latitude, longitude, pressure type, pressure level, temperature, and quality flag. Such a table would repeat the tape ID, synoptic time, profile time, latitude and longitude for all 16 entries in a profile. This redundancy of data is expensive in terms of storage utilization, but the LIMSDATA table does contain all the profile information relating to a given entry record so query results could not fail to provide relevant information although some irrelevant information may be included. An omission of this approach is the ability to provide a sequential access equivalent to that for processing an original FGGE/LIMS data tape. This need exists to permit a minimum of modification to programs which currently process the sequentially organized data tapes, if they were required to access the data through the data base.

To reduce the repetition of data for the entries in a profile, and to solve the sequential access problem, a second set of tables evolved

Table 2-1

Initial Tables for Managing LIMS Data Using ORACLE

TAPE TABLE

TAPE_ID *	SYN_STIME	SYN_ETIME	GENDATE

PRESSURE_TYPE_LEGEND TABLE

CODE *	DESCRIPTION

QUALITY_FLAG_LEGEND TABLE

CHAR *	CODE	DESCRIPTION

LIMSDATA TABLE

TAPE_ID *	SYN_TIME	P_TIME	LAT	LONG	PRESSURE_TYPE	PRESSURE_LVL	TEMP	QC_FLAG

* Denotes Imaged Field

Table 2-2

Tables for Managing LIMS Data Using ORACLE

TAPE TABLE

TAPE_ID *	SYN_STIME	SYN_ETIME	GENDATE

PRESSURE_TYPE_LEGEND TABLE

CODE *	DESCRIPTION

QUALITY_FLAG_LEGEND TABLE

CHAR *	CODE	DESCRIPTION

PROFILE TABLE

TAPE_ID *	SYN_TIME	P_TIME *	LAT *	LONG *

ENTRY TABLE

ROW # *	TAPE_ID	P_TIME	PRESSURE_TYPE	PRESSURE_LVL	TEMP	QC_FLAG

LIMSDATA (VIEW)

TAPE_ID	SYN_TIME	P_TIME	LAT	LONG	PRESSURE_TYPE	PRESSURE_LVL	TEMP	QC_FLAG

* Denotes Imaged Field

(see Table 2-2). The LIMSDATA table was broken into two tables: The PROFILE table containing tape ID, file synoptic time, profile time, latitude and longitude and the ENTRY table containing the row #, tape ID, synoptic time, pressure type, pressure level, temperature, and quality flag. The PROFILE table removes the repetition of synoptic time, latitude and longitude from each entry in a profile. The ENTRY table contains the row # field which is a unique field that application software creates for each row of the ENTRY table. The value is incremented by one and thus represents the sequential order in which the rows were inserted into the data base. If a user accesses the entry table without specifying an ORDER BY or GROUP BY clause, he receives the output in the order in which the data was input to the data base.

To provide the casual user a more friendly interface, a view is defined which looks to the user like the LIMSDATA table in the first set of tables. To produce this effect the view definition joins the PROFILE and ENTRY tables on the common fields tape ID and profile time within each table. A "view" in ORACLE can be used to define what appears to be a table in the data base but which is really some part or parts of one or more existing tables. In this case the view combines information from two tables based on common fields and only omits the artificially produced row # field. The use of this view does result in additional processing overhead since the view definition must be found, appropriate queries to each of the tables must be made and a merged output line must be produced. The view is called LIMSDATA and can be queried using any combination and yielding any combination of the tape ID, synoptic time, profile time, latitude, longitude, pressure type, pressure level, temperature and quality flag fields. A casual user need not know the PROFILE and ENTRY tables exist separately. This further removes the user from having to understand the physical storage of the data. The more sophisticated user could work directly with the PROFILE and ENTRY tables if necessary and, if desired, views could be defined which eliminate the row # field since it has no direct relationship to the data. The major drawback of this approach is the remaining repetition of tape ID and profile time for each row in the entry table. The impact of the row # field should also be noted. Obviously it will add size to each row in the data base, but more importantly this field maintains the sequential order of the data, and, as such, cannot be updated once stored in the table without potentially corrupting the data base. Such a limitation is consistent with the way sequentially organized data sets are managed on magnetic tape (i.e., the observed values cannot be updated or rearranged once on the tape).

To reduce the response delay and repetition of data a third set of tables evolved (see Table 2-3). The ENTRY table no longer contains row #, tape ID, and profile time as before, but a new field, PROFILE_CNT#, has been added which is propagated through all the entries for a given profile. PROFILE_CNT# has been added to the PROFILE table and is an ascending number representing a unique value for each profile in the data base. The synoptic time has now been deleted. These changes result in the same total number of columns in the PROFILE table while a total of two columns have been deleted from each row in the ENTRY table which could contain as many as two million rows.

To provide the user with a friendly interface, a view similar to the LIMSDATA view used in the second set of tables was implemented in this approach as well, except synoptic time is no longer available. The view still requires the joining of the PROFILE and ENTRY tables, but the new view now joins the two with a WHERE clause that matches a row in the PROFILE table with those in the ENTRY table whose value in the PROFILE_CNT# field is equal to the value in the PROFILE_CNT# field of the PROFILE table.
Normally this would result in the matching of about 16 ENTRY table rows with a single row from the PROFILE table. By joining the tables using the PROFILE_CNT# values in each table the sequential access problem is still solved since the responses will be ordered in increasing size of PROFILE_CNT# when a task needs to access all the data the way current software processes data tapes. Since all the fields in the WHERE clause are imaged, all queries using the LIMSDATA view would take advantage of

Table 2-3

Recommended Tables for Managing LIMS Data Using ORACLE

TAPE TABLE

TAPE_ID *	SYN_STIME	SYN_ETIME	GENDATE

PRESSURE_TYPE_LEGEND TABLE

CODE *	DESCRIPTION

QUALITY_FLAG_LEGEND TABLE

CHAR *	CODE	DESCRIPTION

PROFILE TABLE

PROFILE_CNT# *	TAPE_ID	P_TIME *	LAT *	LONG *

ENTRY TABLE

PROFILE_CNT# *	PRESSURE_TYPE	PRESSURE_LVL	TEMP	QC_FLAG

LIMSDATA (VIEW)

TAPE_ID	P_TIME	LAT	LONG	PRESSURE_TYPE	PRESSURE_LVL	TEMP	QC_FLAG

* Denotes Imaged Field

the ORACLE B-trees and reduce access time significantly over the previous approaches. The use of the PROFILE_CNT# fields would be transparent to the casual user who accesses the data base through the LIMSDATA view.

Several general points are worth noting about the approach described above. The use of the view capability requires the joining of tables. Such a process is necessarily slower than if the data is all stored in a single table, but the storage costs of a single table for this application are prohibitive. Secondly, although BTS recommended the use of a single ENTRY table for all eight tapes in this approach it was done in the context of the use of this data base. In general, the production of such a large table is probably undesirable but for the purpose of benchmark testing and because the data base is limited to eight tapes, the data base has been so designed.

2.1.2.1.2 Estimation of ORACLE Data Base Storage Requirements

In estimating the storage requirements for the data base design in Table 2-3, it is possible to ignore the size of the three small tables and only consider the ENTRY and PROFILE tables. In making the size estimate the calculations are based on the available FGGE/LIMS data which amounted to two million rows of data in the ENTRY table (which corresponds to two million input records in this case). The first step in estimating the data base size is to calculate the average row size. The FGGE/LIMS data base has no null fields so each domain will contain two bytes of overhead plus a variable number of bytes for the actual data representation. The field size estimates are shown in Table 2-4. The 2.2 version of ORACLE manages disk storage in 64-byte groups and, as a result, must write each row inserted into the data base in a multiple of 64 bytes. Table 2-4 shows the calculations of the row sizes for both tables. The estimated sizes indicate that 30 and 42 bytes are wasted as overhead in each row of the PROFILE and ENTRY tables, respectively, due to the current method of disk memory management used by ORACLE. A word level (16 bits) management capability is planned by RSI in the future, possibly in the 3.0 version to be released in late 1981.

Table 2-4
Table Size Estimates

Profile Table

Domain	Data Type	Average Size*
PROFILE_CNT#	Numeric	6
TAPE_ID	Character	8
P_TIME	Character	12
LAT	Numeric	4
LONG	Numeric	4
TOTAL ROW SIZE		34 Bytes

# of Rows * Minimum Row Size** = Table Size (Bytes)
130,000 Profile Rows * 64 = 8,320,000.

Entry Table

Domain	Data Type	Average Size*
PROFILE_CNT#	Numeric	6
PRESSURE_TYPE	Numeric	4
PRESSURE_LVL	Numeric	4
TEMP	Numeric	4
QC_FLAG	Numeric	4
TOTAL ROW SIZE		22 Bytes

# of Rows * Minimum Row Size** = Table Size (Bytes)
2,000,000 Entry Rows * 64 = 128,000,000.

* Includes Overhead Bytes
** ORACLE Version 2.2 must manage rows in multiples of 64 bytes. RSI promises to improve this in the near future to a 2 byte capability.

An estimate of the index space required for the ORACLE B-trees is more difficult to make. RSI has provided relatively little formal documentation about the internal B-tree algorithms and structure but through their representative and some experimentation the following was learned:

- The B-tree leaves and nodes are 512 byte blocks of data
- The B-tree leaves can be estimated to be 75% utilized since they are divided in half when they become full.
- That because of compression techniques a leaf block may be considered, for estimation purposes, to hold 50 keys when full or approximately 37.5 keys when 75% full.
- That the node blocks could also be assumed to be at 75% utilization.
- That for each 37.5 leaf blocks a node block would be required at the 75% utilization point.
- That 512 blocks are reserved for user defined views regardless of data base size.
- That 300 blocks are reserved for data dictionary use regardless of data base size.

By using the above guidelines an estimate of data base size could be made. A small FGGE/LIMS data base that was full and whose row numbers and keys were known provided evidence that the guidelines were very acceptable for this application.

Dividing the total number of keyed values by 37.5 should approximate the number of leaf blocks. To obtain the number of node blocks it is

Table 2-5

Index (B-Tree) Size Estimate

TABLE NAME	ROWS	# OF KEYS/ROW =	# OF KEYS
PROFILE	130,000	4	520,000
ENTRY	2,000,000	1	2,000,000
TOTAL NUMBER OF KEYS IN DATA BASE			2,520,000

Leaf Calculation -

(Assume leaf blocks 75% full and contain 37.5 keys/block)

2,520,000 keys   37.5 keys/block * 512 bytes/block = 34,406,400 bytes

Node Calculations -

(Assume node blocks 75% full and contain 37.5 pointers/block)

(34,406,400 Leaf Bytes   512 bytes/block = 67,200 leaf blocks)

67,200 leaf blocks   37.5 pointers/block = 1,792 1st level node block

1,792 1st level nodes   37.5 pointers/block = 48 2nd level node block

Plus a single root block = 1 root node block

TOTAL NUMBER OF NODE BYTES = 1,841 * 512 = 942,592

Leaf Bytes	34,406,400
Node Bytes	942,592

TOTAL INDEX SIZE 35,348,992 BYTES

Table 2-6

ORACLE Data Base Size Estimate*

ITEM	SIZE IN BYTES
Reserved Space for User Views (Independent of Data Base Size)	262,144
Reserved for Data Dictionary Use (Independent of Data Base Size)	153,600
PROFILE Table (See Table 2-4)	8,320,000
ENTRY Table (See Table 2-4)	128,000,000
Index Requirements (See Table 2-5)	35,348,992
ESTIMATED SIZE OF DATA BASE	172,084,736

*Estimate is for a data base containing two million entry records and the associated profile information.

assumed that for every 37.5 leaf blocks a node block is required, and for every 37.5 node blocks another level of node blocks is required until the root block is reached. The need for estimating or sizing the data base is crucial and the lack of RSI-supplied documentation on this subject at this time is disappointing. Table 2-5 details the estimation procedure for predicting the B-Tree storage requirements for the FGGE/LIMS application.

Table 2-6 summarizes the estimates for each of the significant contributors to the data base size. The bottom line figure shows that slightly more than 172 million bytes are estimated to be required to store 2,000,000 rows of entry level data plus 130,000 rows of profile level data using the 2.2 version of ORACLE. Noteworthy is the fact that about 88 million bytes of storage is dedicated to the wasted overhead created by the 64 byte storage management now employed. A future change to a word level (2 byte) management approach would eliminate this waste but would increase the number of bit maps required for managing the data base by a factor of 32 and would add processing overhead as well.

2.1.2.2 SEED Data Base Design for Satellite Data

2.1.2.2.1 SEED Data Base Construction

Figure 2.1 in conjunction with Table 2-7 describes a SEED data base structure for FGGE/LIMS data. The structure which resides in a single area is very simple and straightforward. A single tape of FGGE/LIMS data is represented by an occurrence of the CALC record R3_TAPEID. Each tape will have a synoptic time range associated with it represented by the owner occurrences of the records R1_SYN_STIME, which contains the synoptic start time, and R2_SYN_ETIME, which contains the synoptic end time. The R4_DATATYPE record indicates the type of data on the tape, in this case FGGE/LIMS and corresponds loosely to the small tape table. That is, one could query for all tapes which have FGGE/LIMS data. This record-type could have been omitted for the testing but its inclusion does not contribute significant overhead to the results.

FIGURE 2.1
SEED FGGE/LIMS CONCEPTUAL
DATA BASE

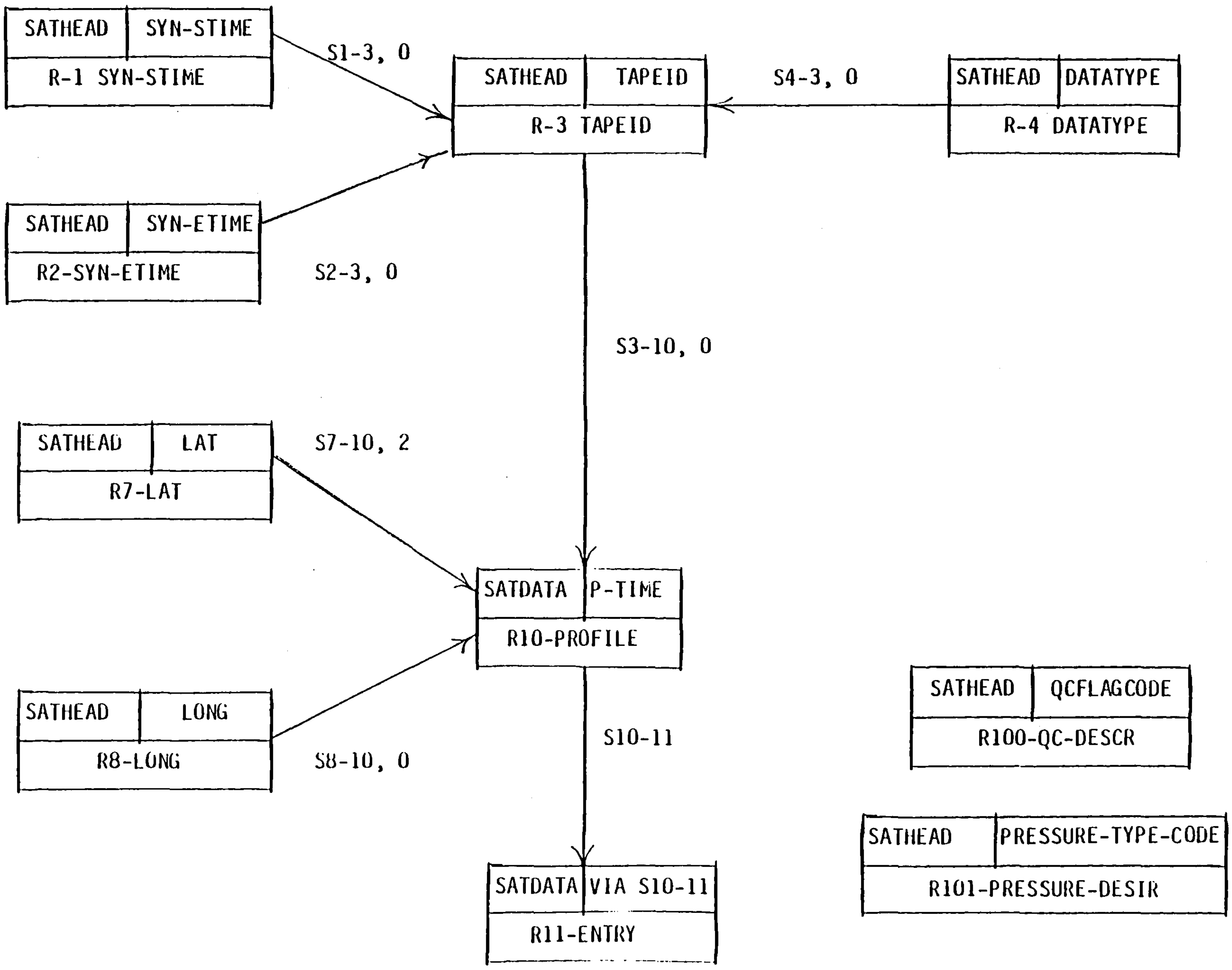


Table 2-7

SEED Record Description and Storage Requirements

Record Name	Field Name	Type	Data Storage/ Rec	Set Linkage Overhead/ Rec
R1-SYN-STIME	SYN-STIME	CHAR*8	8	8
R2-SYN-ETIME	SYN-ETIME	CHAR*8	8	8
R3-TAPEID	TAPEID GENDATE	CHAR*6 CHAR*6	12	32
R4-DATATYPE	DATATYPE	CHAR*30	30	8
R7-LAT	LAT	INTEGER*4	4	8
R8-LONG	LONG	INTEGER*4	4	8
R10-PROFILE	P-TIME	CHAR*10	10	32
R11-ENTRY	PRESSURE-TYPE PRESSURE-LVL TMP QC-FLAG	INTEGER*2 INTEGER*4 INTEGER*2 INTEGER*2	10	4
R100-QC-DESCR	QC-FLAG-CODE QC-DESC	INTEGER*2 CHAR*60	64	4
R101-PRESSURE-DESCR	PRESSURE-TYPE-CODE PRESSURE-DESCR	INTEGER*2 CHAR*60	64	4

A tape consists of many profiles and each profile consists of several entries. This relationship (one to many) is established by the use of the R10-PROFILE record which is a CALC record on profile time (P-TIME) and the R11-ENTRY record which contains detail data on each profile. Table 2-7 lists the contents and structures of the data base records. Also associated with each profile is a latitude and longitude. These are represented by the R7-LAT and R8-LONG records respectively. Note that the structure illustrated in Figure 2.1 allows an R7-LAT or an R8-LONG record to own many different profiles. This situation occurs quite often as there are many profiles at a particular latitude and longitude. This structure should reduce query time and eliminate duplication of data.

The CODASYL data structure depicted in Figure 2.1 shows the relationships of the FGGE/LIMS data base. The structure is depicted as boxes which correspond to record types and arrows which correspond to set types. Each set type has a record declared as the owner record (which would be at the tail of the arrow) and a record declared as the member record (which would appear at the head of the arrow). Appendix II, Part B, contains the Data Description Language (DDL) that is used to define the schema for this data base application.

2.1.2.2.2 Estimation of SEED Data Base Storage Requirements

The size of the SEED data base must be defined prior to loading the data base by specifying its constituent sizes in the schema. In making an estimation of the data base size one must first approximate the total number of records to be managed by the data base. For the purpose of this estimation it is assumed that there would be a maximum of 130,000 PROFILE records and 2,000,000 ENTRY records. The schema specified that the ENTRY records were to be members of a "VIA" set owned by PROFILE records. This means that the ENTRY records associated with a PROFILE time are loaded physically as close to each other as possible (i.e. on the same page or an overflow page if this page is full). A single area was selected to contain this information. The pages in this area were defined to be 1024 bytes long and a maximum of 38 records per page was specified. (The figure 38 was originally selected when it was thought that a single page would always contain all member records for the associated owner record. This was found to be an incorrect assumption only after the testing had begun. The impact of this design error is to decrease the density of the data on a page.) Table 2-7A describes the calculations used to determine the size of the area that contained the profile time values and the ENTRY records. The record sizes were estimated by calculating space required for data, set linkages (chain pointers), and overhead. Two logical areas of the data base were defined in the schema. One was the SATHEAD area which contained the latitude and longitude values associated with a profile time as well as the tape-id and data-type occurrences. The other was the SATDATA area which contained the profile times and all the entry data.

Because of the VIA set storage approach used, an average number of PROFILE and ENTRY records could be determined based on their ratios and the maximum number of records per page. Multiplying the size of each record type by the average occurence per page and summing the values yields a result of 685 bytes per page. The remaining 339 bytes on each page will not be utilized because of the limiting specification of 38 records per page. A total number of pages is determined by dividing the total number of PROFILES to be entered into the data base by the average number of PROFILES per page. The resulting figure multiplied by the bytes per page figure yields a required SATDATA area size of 59,164,672 bytes. It is important to remember that of this amount only 39,577,930 would be utilized. The remainder is unused due to the schema design.

The SATHEAD area estimate is shown in Table 2-7B. Because of the small number of occurrences of some of the records their space considerations are ignored. The LAT and LONG records are the only significant records to consider. Both are calculated to have record sizes of 14 bytes. The page size is 512 bytes in the area so the maximum of 38

TABLE 2-7A
CALCULATION OF SATDATA AREA SIZE
ASSUMING 130,000 PROFILE RECORDS

Maximum Number of Records/Page = 38   Page Size = 1024 Bytes

Page Averages:	2.25 Profile Records
	35.75 Entry Records
	38 Records/Page

Profile Record Size	=	10 Bytes Data
		32 Bytes Set Linkage
		2 Bytes Record Header
		44 Bytes/Record
Entry Record Size	=	10 Bytes Data
		4 Bytes Set Linkage
		2 Bytes Record Header
		16 Bytes/Record
Page Utilization	=	14 Bytes Page Overhead
		99 Bytes Profile Data (2.25 Record * 44 bytes record)
		572 Bytes Entry Data (35.75 Record * 16 bytes record)
		685 Bytes Per Page

Unused space/page due to design	=	339	Bytes (1024 bytes per page-685 bytes utilized)
number of pages in area	=	57,778	(130,000 profiles 2.25 profiles/page)
SATDATA area	=	59,164,672	Bytes (57,778 pages * 1,024 bytes/page)
SATDATA area utilized	=	39,577,930	Bytes (57,778 * 685 bytes/page)

TABLE 2-7B
CALCULATION OF SATDATA AREA SIZE

Maximum Number of Records Page = 38   Page Size = 512

SATHEAD would contain a maximum of:

1	DATATYPE record
8	SYN_STIME records
8	SYN_ESTIME records
8	TAPE_ID records

(For calculation purposes the above records are ignored due to their numbers)

18001	LAT records (values range from - 9000 to 9000)
36001	LONG records (values range from 0 to 36000)

LAT Record Size	=	4	Bytes Data
		8	Bytes Set Linkage
		2	Bytes Record Overhead
		14	Bytes/Record
LONG Record Size	=	4	Bytes Data
		8	Bytes Set Linkage
		2	Bytes Records Overhead
		14	Bytes/Record

number of records per page	=	36	(512 bytes per page ÷ 14 bytes/record)
Maximum number of records	=	54,002	(18001 LAT records + 36001 LONG records)
number of pages required	=	1,501	(54002 records ÷ 36 records/page)
SATDATA area	=	768,512	Bytes (1501 pages * 512 bytes/page)

records is not reached because at 14 bytes per record only 36 records can be stored on a page. The LAT and LONG records can assume a maximum of 54,002 possible different values so this is the constraining limit rather than the 130,000 PROFILES they are related to. By dividing the possible values by the number of records per page, the number of pages required in the area is obtained. The total bytes required would be 768,512 bytes of which virtually all space is utilized. Summing the utilized space in each area reveals a space requirement of 40,349,442 bytes.

2.1.2.3 RIM Data Base Design for Satellite Data

2.1.2.3.1 RIM Data Base Construction

The RIM DBMS is based on the relational algebra model and, as such, is similar to ORACLE. The RIM package was introduced to the study just prior to beginning the quantitative benchmarking. Ample time did not exist to experiment with the package to determine what special attributes or limitations it might have, but there appeared to be no problem using a data base design identical to the ORACLE design. Logically, this seemed to be the wisest approach based on the lack of experience with the package and the desire to provide a test bed from which analogous measurements could be made. All testing was performed on version 4.0 of RIM.

After preliminary testing it was determined that a RIM data base was made up of three files. The first file appeared to contain the information that defined the tables, domains, and other relevant information about the data base structure and relationships. The second appeared to contain the actual data in the data base. The third, and last file, appeared to contain the inverted files or B-trees responsible for managing the data base indices.

As a result of some initial loads of the LIMS data, it was determined that one deviation from the ORACLE design was mandatory. The profile time was an indexed domain in the ORACLE Profile Table. The time was represented as a ten-character string (as in the ORACLE and SEED designs) because it was too large to be numerically represented. Initial loads indicated that the managing of this field when indexed resulted in significant degradation in RIM load and query performance (see Section 2.2.4) and, as a result, this field was redefined as non-indexed for the benchmarking effort. In comparing the benchmark results it must be taken into account that the RIM load rates reflect the indexing of one less indexed field after the 99,000 record level and that a different query was used in testing indexing efficiency. The exact impact of these changes is impossible to determine at this time but the steps taken were necessary to continue the testing.

2.1.2.3.2 Estimation of RIM Data Base Storage Requirements

Lack of documentation regarding internal storage methods and overhead in RIM prevented preliminary estimates of data base size as were made for the ORACLE and SEED systems. The absence of internal data base documentation needs to be corrected but the lack of this information was not too serious due to RIM's dynamic approach to allocating space. (RIM dynamically requests more space for its files as the data base grows.) Thus, a preliminary estimate of data base size was not required, though this was necessary for SEED and ORACLE. Obviously, a production DBMS must provide a methodology for estimating data base size so that storage media requirements or availability may be determined.

2.2 Loading the Data Bases

2.2.1 General Approach to Loading

The LIMS satellite data designated for use in ORACLE, RIM and SEED data bases was available on eight magnetic tapes. As already stated, these tapes included approximately 130,000 profiles and approximately 2,000,000 records associated with the profiles. Initially it was considered desirable for load software to be designed and coded which could insert data into the data bases directly from the LIMS tapes. A high level design was formulated for a main or "driver" routine that could call routines to read and select data from the LIMS data tapes and call sets of load-specific routines written to load the data base structures managed by either ORACLE, SEED, or RIM.

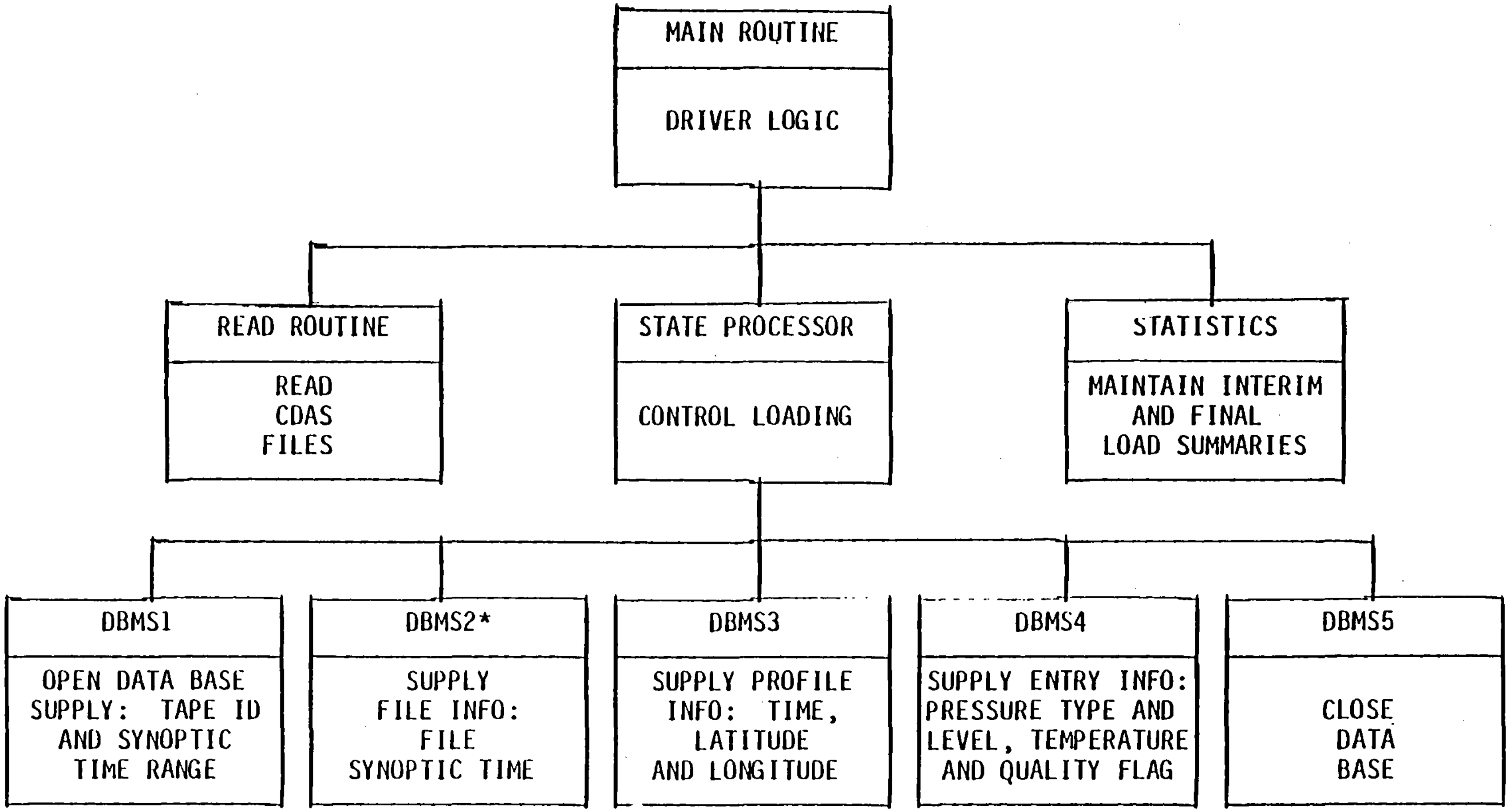
This approach would minimize bias in evaluating the loading capabilities of SEED, ORACLE, or RIM since all sets of load software would be interfaced with the same logic for reading the tapes and supplying data to be inserted into the data bases. The common logic was designed to make five calls to the load software. The first call would pass the tape id (obtained by prompting the user, since tape id is not available on a LIMS tape), synoptic start and stop time, and the tape generation date. The second would be made for each file on the tape and would identify the synoptic time of the file. The third would be made for each profile and would identify the profiles actual time to the minute, the latitude, and the longitude. The fourth would pass a set of entry values for a given profile by passing a number of pressure type, pressure level, temperature, and quality flag values. The fifth call would signal the end of data on the tape and should result in the closing of the data base.

The implementation of this approach uncovered several problems which resulted in some alterations. All the software was to be coded in FORTRAN but it was found that VAX FORTRAN I/O could not be used to read the LIMS data tapes. Rather than spend time implementing routines using the VMS QIO capability it was determined that files created by the Climate Data Access System (CDAS) could be read easily using VAX FORTRAN I/O. Because of the urgency associated with loading the satellite data the main routine was modified to read disk files containing the FGGE/LIMS data that had been generated by CDAS. This indirect approach would ordinarily be undesirable, but time limitations required taking such a shortcut. The only drawback of the CDAS version of the LIMS data is a loss of the file synoptic time, but this omission within the data bases is of little or no significance. Each set of load software was modified to ignore the call to the DBMS2 routine (see Figure 2.2) for file synoptic time and the main routine was set up to make only one such call during a run. A functional description of the load software appears in Figure 2.2. An additional capability provided for the load software was logic to provide measurements of load rates on an interim and summary basis.

The benchmark tests were to be conducted at various levels of data base size. The actual levels chosen were in part a function of the amounts of data on the FGGE/LIMS data tapes but were intended to reveal sensitivities of performance to diferent volumes of information. The actual levels of data base size chosen (expressed in terms of the sum of Entry and Profile records read off the CDAS files) were: 52,000, 99,000, 187,000, 439,000 and 1,039,000. Although the original plan suggested reaching a 2,000,000 record level, time and disk storage limitations forced a cut off at approximately 1,000,000.

The load rates were measured by reporting elapsed "wall clock" time, elapsed CPU time, total direct I/O's and total page faults as reported by VAX system monitor routines for the load software routines. These statistics' with the exception of "wall clock" time, do not reflect the majority of the processing overhead in ORACLE's case because a detached process is created which does much of the actual processing and which is not included in the reported figures directly (see Sections 2.2.2 and 2.2.3). The measurements were reported for each 5000 successive Profile and Entry records inserted into the data base from the CDAS file. From these measurements one can determine the relative insertion rate at any particular level of data base size and thereby draw conclusions about degradation or efficiencies.

To produce results that were comparable between incremental loads or across DBMS's, dedicated VAX system time was desirable. Due to the amount of such time required the loads could not be conducted in a totally stand-alone environment. The loads were all conducted from some time after 6:30 p.m. and almost all were completed prior to 8:00 a.m. During the first half hour of each load the system was reserved for stand-alone


MAIN ROUTINE
DRIVER LOGIC
READ ROUTINE
READ CDAS FILES
STATE PROCESSOR
CONTROL LOADING
STATISTICS
MAINTAIN INTERIM AND FINAL LOAD SUMMARIES
DBMS1
OPEN DATA BASE SUPPLY: TAPE ID AND SYNOPTIC TIME RANGE
DBMS2*
SUPPLY FILE INFO: FILE SYNOPTIC TIME
DBMS3
SUPPLY PROFILE INFO: TIME, LATITUDE AND LONGITUDE
DBMS4
SUPPLY ENTRY INFO: PRESSURE TYPE AND LEVEL, TEMPERATURE AND QUALITY FLAG
DBMS5
CLOSE DATA BASE

*DBMS2 is only called once and the DBMS2 load routines simply return when called due to the omission of file level data in the CDAS data files.

Figure 2.2
Functional Description of Load Logic

loading to provide controlled results. This meant that other users could have logged on the VAX after the first half hour of a load and could have performed operations which contend with the load routines for CPU resources. However examination of the load results and informal monitoring of terminal room activity by benchmarking personnel and VAX personnel indicate that contention was rarely encountered. With this in mind one may use the measurements to determine the relative degradation in load rates, if any, due to data base size to compare performance between systems. The actual numbers generated must be viewed in the context of the data base design and the isolated environment they were produced in. A design with more fields or more indexes would likely degrade lead performance. The introduction of other VAX users could also affect load rates but predicting how is dependent on the system "mix", the contending processes and their associated quotas and priorities, as well as the load routines' associated quotas and priorities. Clearly empirical testing is required to estimate rates in a particular environment and the figures produced in the benchmark loads would normally have to be considered best case or "top end" figures.

2.2.2 ORACLE Load Results

The ORACLE load process required that at the beginning of each load that a query be made of the data base to determine the maximum value of the PROFILE_CNT field which was artificially created and added to the data (see Section 2.1.2.1.1). The overhead incurred by this query is small initially but increases as more rows are added to the Profile table. Also at the beginning of each load is the overhead of logging onto the data base and opening it. The net effect of the overhead is an apparent degradation in the initial 5000 rows inserted into the data base at each incremental load step. Accompanying this discussion are a number of graphs depicting load performance. The points plotted on these graphs represent the "wall clock" time required to load the CDAS records into the data base at a particular data base size. The points are determined by dividing the wall clock time since the last 5000 record measurement into 5000 to get the number of records inserted per second. This provides an insertion rate per second averaged over the last 5000 entries (approximately 300 Profile records and 4700 entry records). The resulting pictorial summary of load performance should demonstrate any sensitivities to data base size or particular external conditions such as user contention.

During the initial loads using Version 2.2 of ORACLE the data base was defined with relatively small data base "extends" that were sufficient for the amount of data in the data base at that time. An ORACLE "extent" is an addition to the data base made after it is originally defined and initialized. Extents normally allow the data base to grow gracefully without requiring one to reserve large amounts of space prior to needing them. An extent is the equivalent of a VAX disk file. After reaching the 180,000 row level the number of "extents" was increased to a total of six in anticipation of going to the 439,000 row level. When that load was begun the detached process exited before the completion of the load. The original load process remained in what appeared to be a dormant state. It was then cancelled by benchmark personnel. Subsequent examination of the data base revealed about 220,000 rows were then present. Discussion of the problem with RSI personnel indicated that a problem existed in the ORACLE software when a large number of data base "extents" were defined. RSI suggested reinitializing the data base to a single large extent and reloading the data. This was done and a new set of loads were started. Approximately 290,000 rows were inserted into the data base before a similar failure occurred. That is, the detached process exited leaving the parent load process in a semi-dormant state. An attempt to restart the load was futile. Further consultation with RSI revealed a problem with the 2.2 version in managing more than 65,000 blocks of memory. A patch was provided for the ORACLE software but it proved ineffective and RSI agreed to provide a preliminary copy of Version 2.3 that would solve the management problems. The results of these original loads are not presented in any of the graphs provided because they are superseded by those from Version 2.3.

Before discussing the Version 2.3 load rates, some discussion of the ORACLE methodology used for processing is worthwhile, especially in light of our earlier load failures. When software "logs on" to ORACLE to begin the load, ORACLE creates a detached process that appears to perform most of the work required during the session. RSI documentation does not detail the methodologies employed to communicate with the detached process but problems seem to exist. When the detached process encounters a fatal error and must prematurely terminate, the original load software is left uninformed. This is a very serious problem because the user has no way of knowing his session has essentially stopped nor does he have any way of finding out the actual cause of the original failure other than a "post mortem dump". The significance of this shortcoming is apparent and should be corrected. The graphs provided depict results of successful load rates only.

The ORACLE load rates produced using VERSION 2.3 are extremely uniform. Figure 2-3, Part 1, shows rates plotted for the initial 100,000 rows loaded into the data base. Note should be made that the degradation at about 58,000 rows is probably associated with the overhead of the initial maximum Profile count query mentioned earlier. The degradation over the first 100,000 rows is barely detectable dropping from around 7.4 rows/second to around 7.2 rows/second. The 2.3 version of ORACLE appears to be a bit slower in load performance than Version 2.2 (whose results are not provided here). This may be caused by the introduction of the "journaling capability" but whatever the cause, the load rate was reduced about 12%. Figure 2.3, Parts 1 and 2 shows the load rates from the 100,000 row level to the 400,000 row level. The load rate remains very uniform with a very slight degree of degradation. Over this 300,000 range the insertion rate drops less than .4 rows/second going from slightly below 7.2 rows/second to about 6.8 rows/second. Figure 2-3, Parts 3, 4, and 5 depicts the load rates from the 400,000 row level to 1,039,000 rows. The results continue to display uniformity with very slight degradation. After loading over 600,000 records the load rate had dropped to a final rate of about 6.2 rows/second. Figure 2.4 summarizes the results of the entire load on a single page and shows the consistency of the load rates.

At the end of the first load Figure 2-3 Part 1 displays a noticeable dip which is unexplainable at this time. Examination of the last observation point of the succeeding loads shows no such behavior. The breaks in the graph represent the pauses in the load sequence where a new load was begun. The total degradation in insertion rate from 7.4 to 6.2 rows per second is most likely attributable to the size of the B-trees required for the indexed fields. The very slight irregularities that exist along the plotted points are probably attributable to random efficiencies and costs due to the physical location of the disks when reading from the CDAS files and writing to the data base. (The input files were located on a different device than the output devices but they were both associated with the same controller as was the case with the SEED and RIM applications).

Examination of both the load software process and the detached process associated with the load show that their cumulative CPU utilization in an uncontested environment was around 77% of available CPU time. The ratio of the load software's utilization to the detached process's utilization is about 1 to 4.5. The detached process does most of the processing associated with the insertion procedure which is demonstrated by the proceeding ratio. The ORACLE DBF LU command shows that the final data base containing about 1,039,000 record requires 169,142 blocks or about 87 million bytes. This supports the methodology applied for estimating the ORACLE data base size in Section 2.1.2.1.2 which predicted 172 million bytes would be required to manage two million rows. The actual number of rows in the data base is 52% of the two million rows and has consumed about 51% of the space originally estimated for it.

2.2.3 SEED Load Results

The approach used for the managment of Profile and Entry information was partially described in Section 2.1.2.2.2. The implication of the

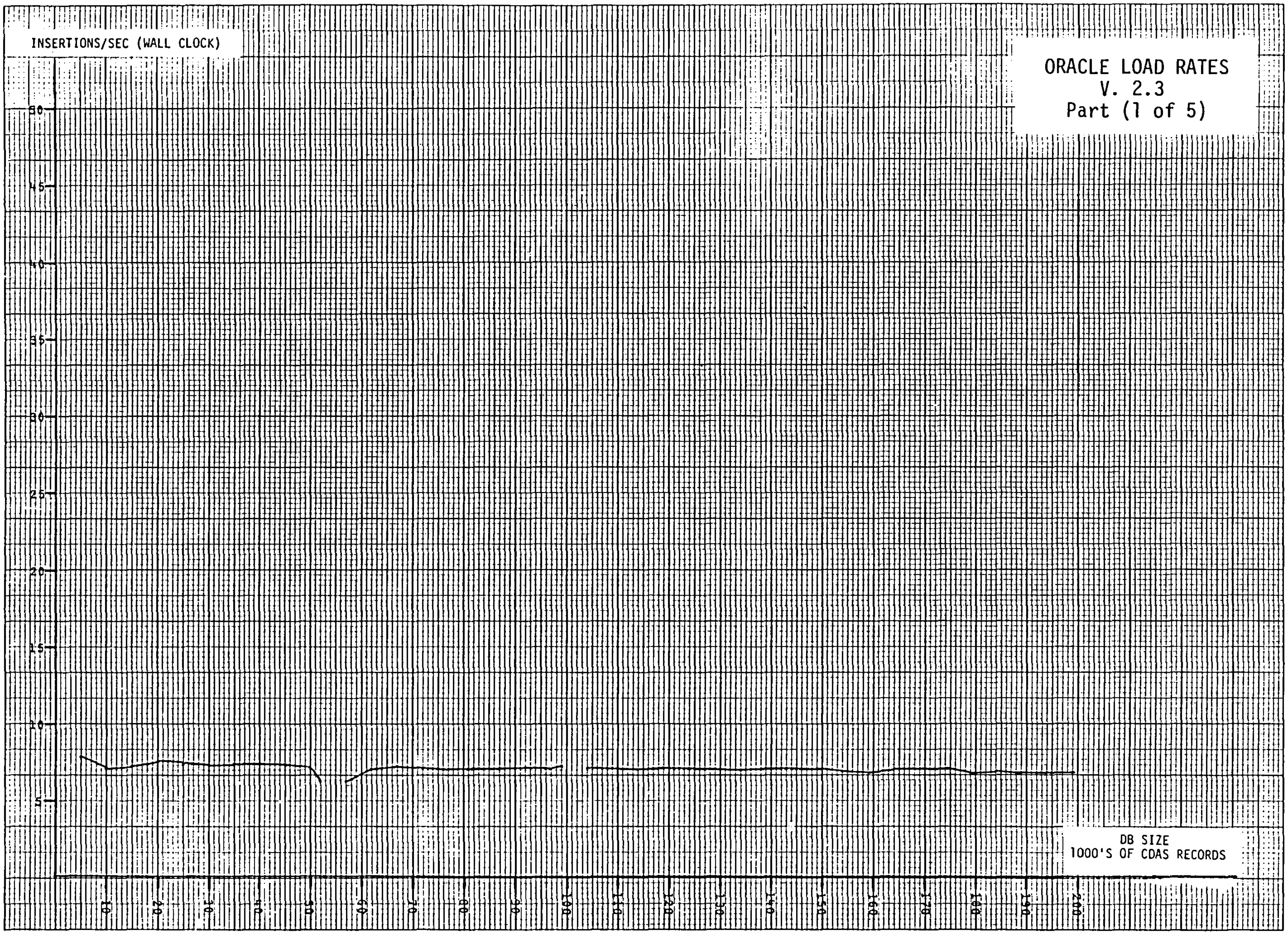


Figure 2.3, Part 1

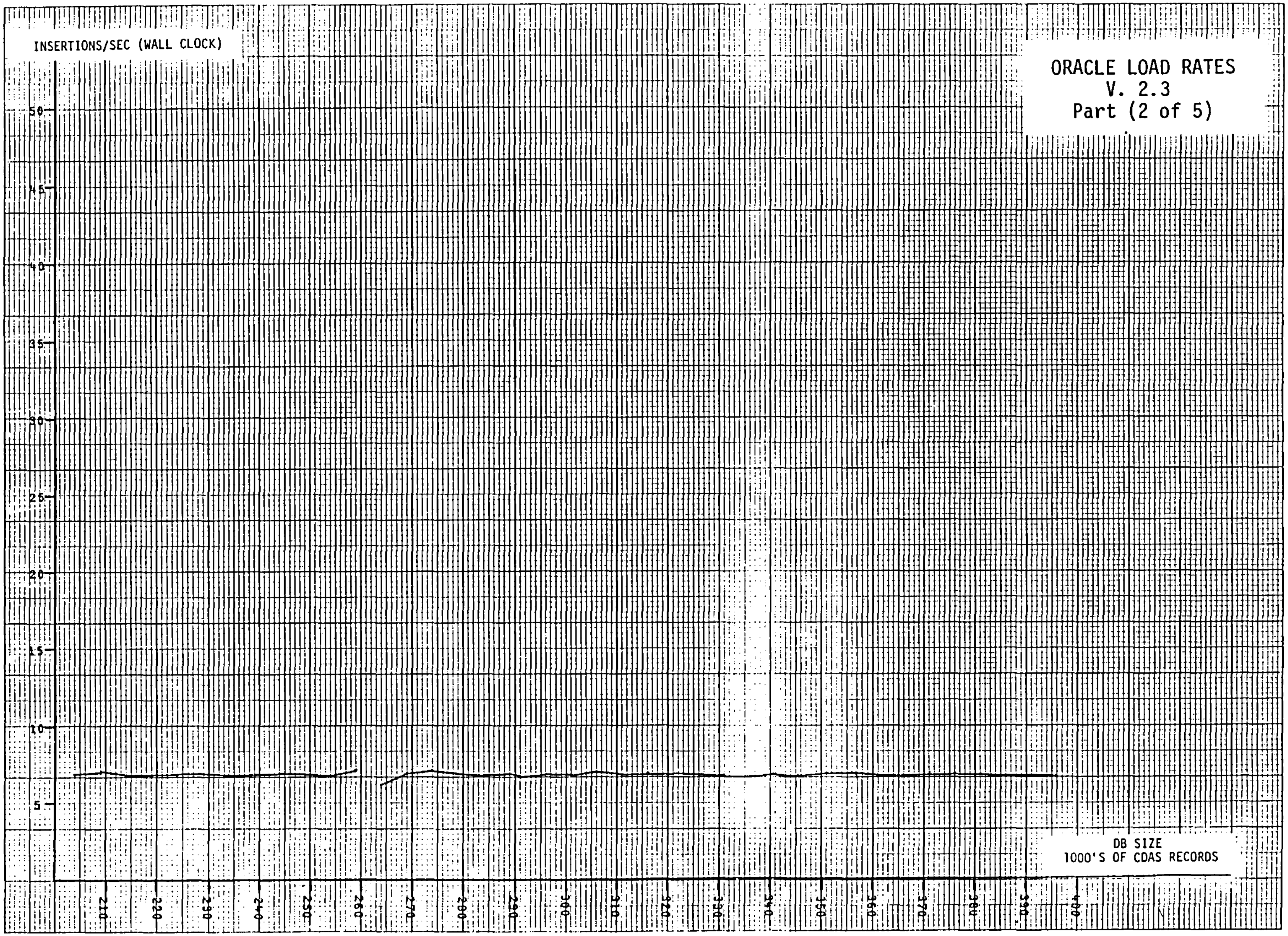


Figure 2.3, Part 2

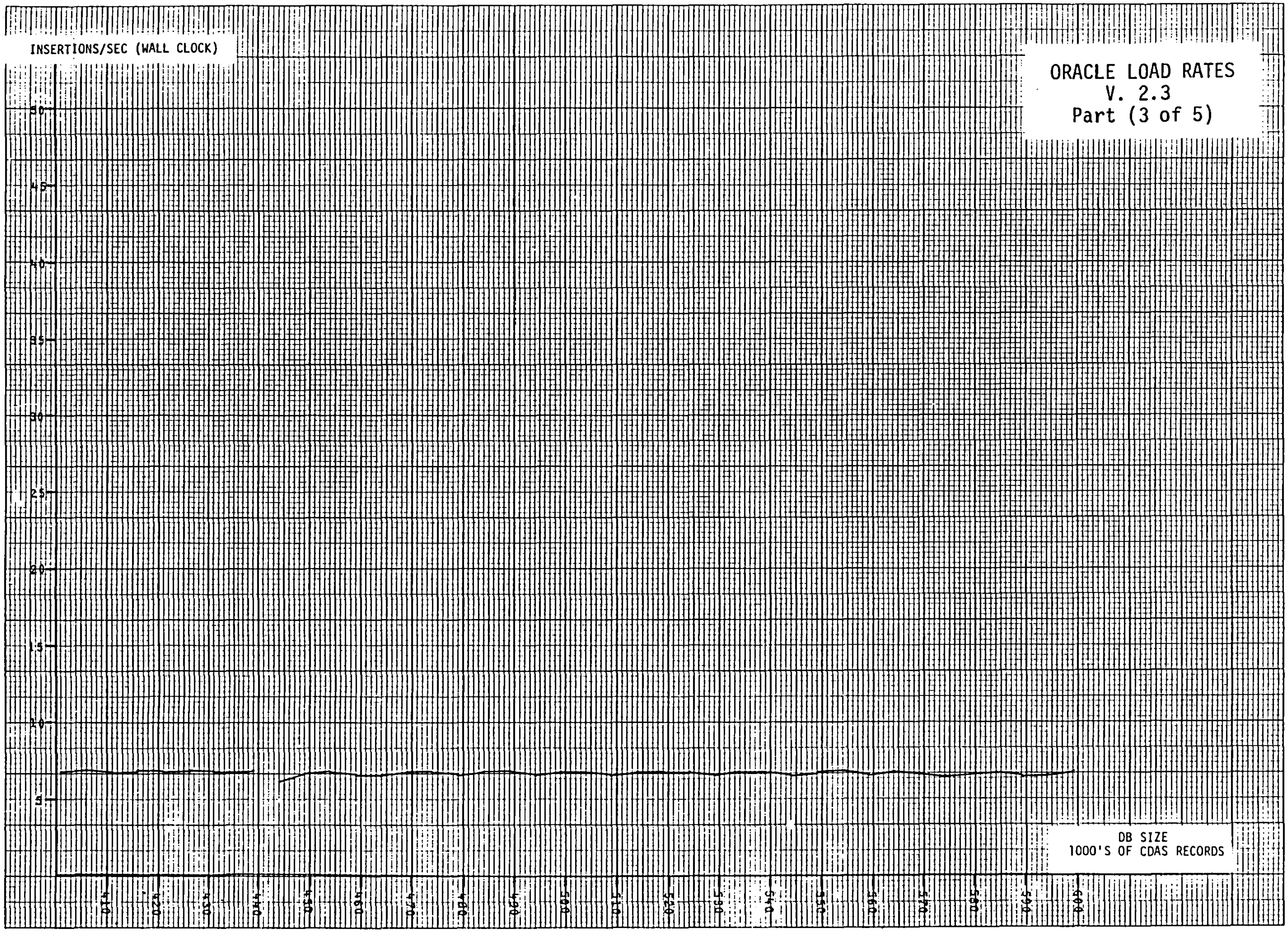


Figure 2.3, Part 3

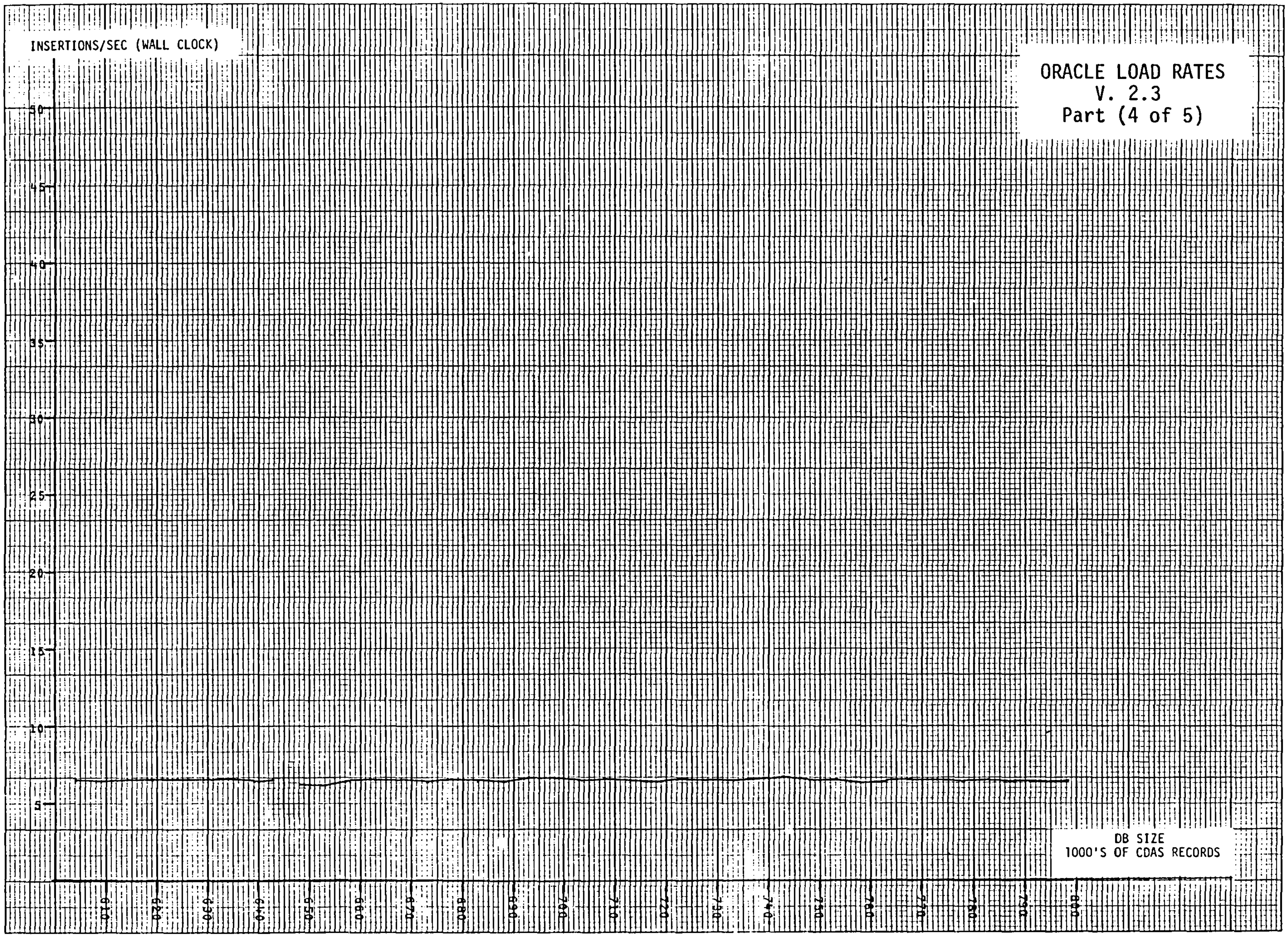


Figure 2.3, Part 4

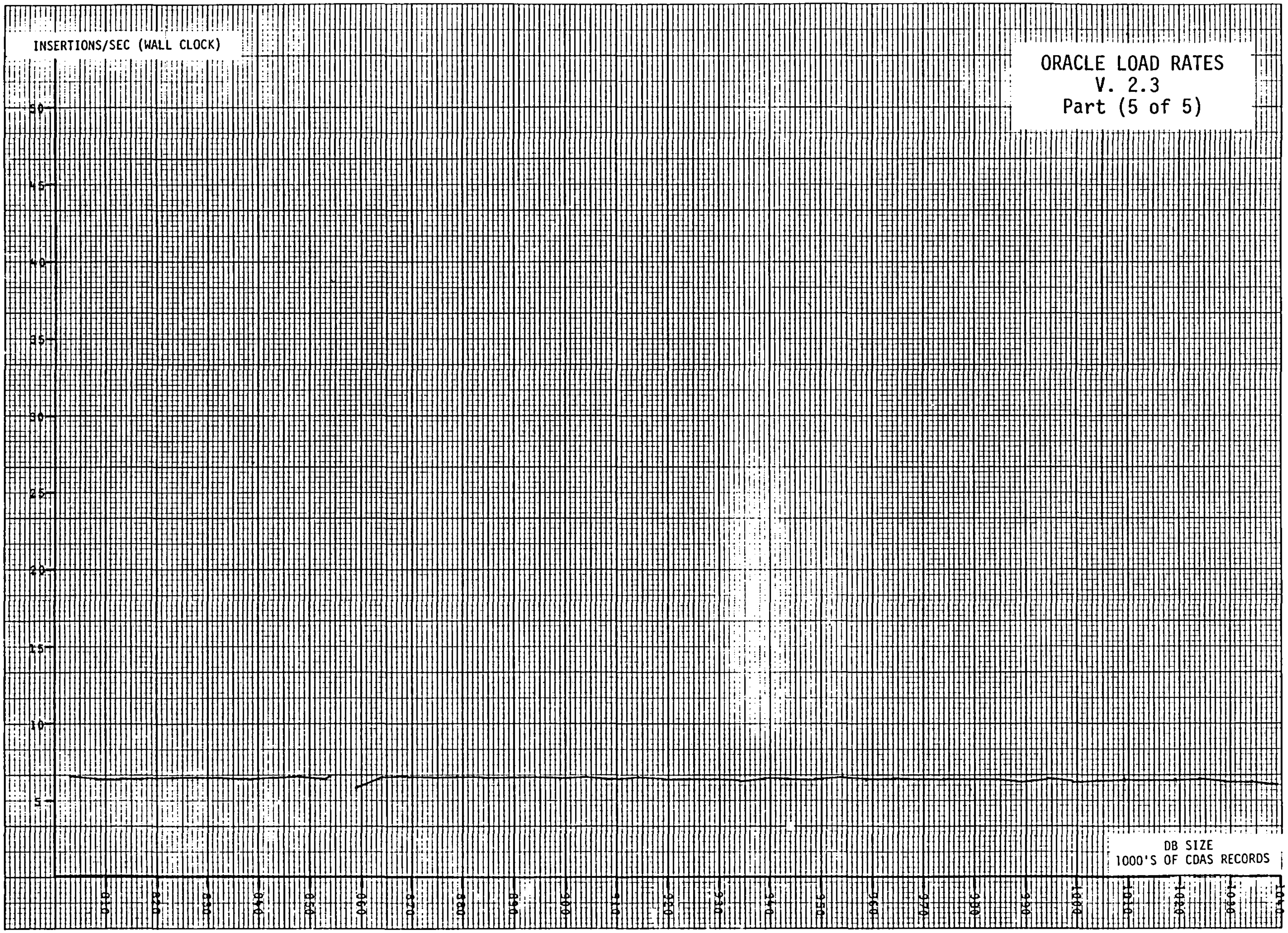


Figure 2.3, Part 5

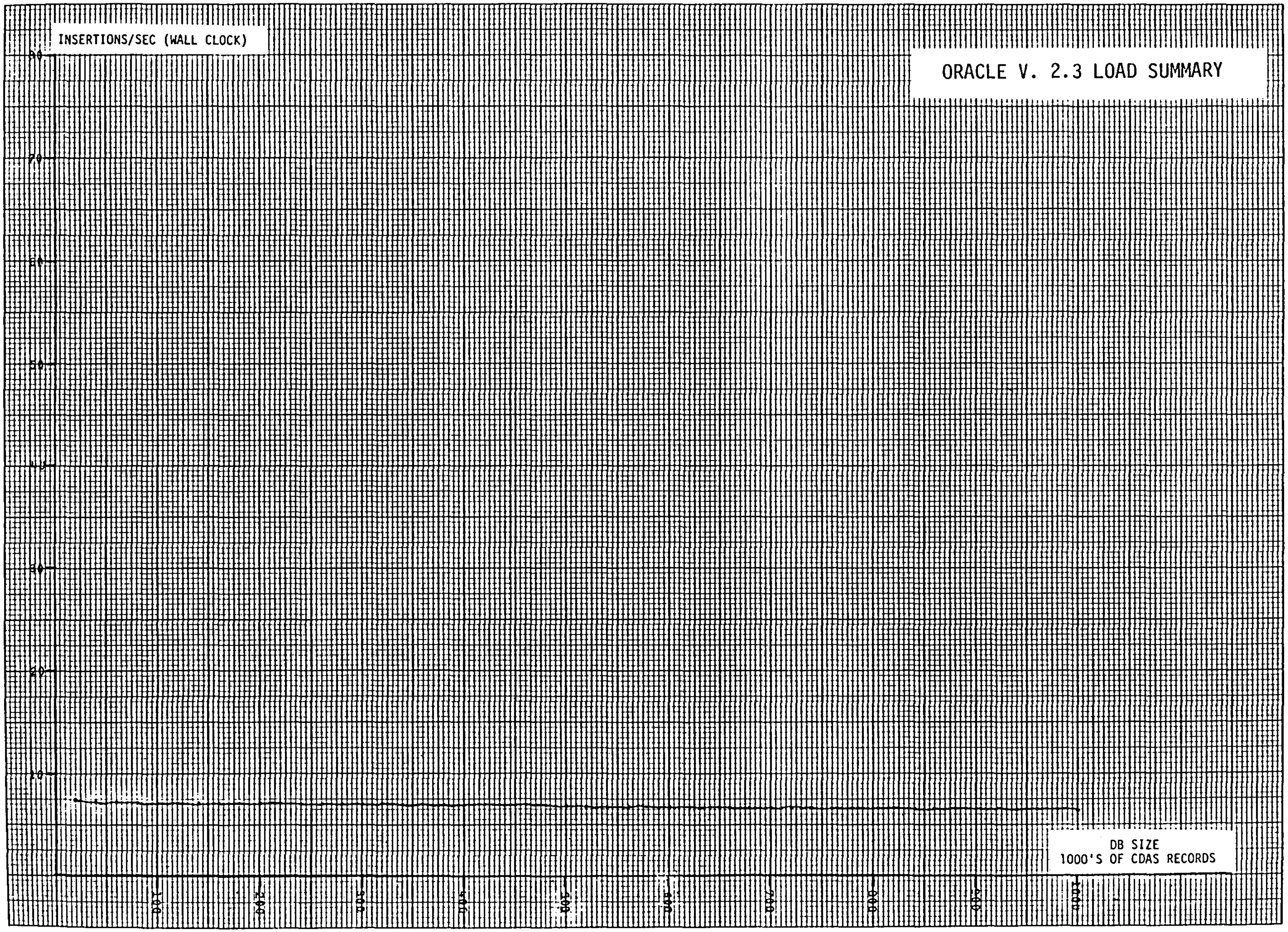

INSERTIONS/SEC (WALL CLOCK)
ORACLE V. 2.3 LOAD SUMMARY
80
70
60
50
40
30
20
10
DB SIZE
1000'S OF CDAS RECORDS
100
200
300
400
500
600
700
800
900
1000

Figure 2.4

description is that a Profile and the Entry information related to it would be physically stored as close to each other as possible. By defining the Profile and Entry data relationship as a "VIA SET" to SEED this physical adjacency was accomplished. All of the benchmark testing was conducted with this approach, however, subsequent to the testing several test loads were performed using a "DIRECT" loading approach which isolated the Profile and Entry information into separate areas. The results of those loads are summarized at the end of this section but no query tests were made for the data so loaded.

The initial SEED loads were conducted using Version B11.2 with a data base defined as "dynamic" which would allow the data base to grow when overflow conditions occurred. The load rates associated with the first 99,000 records are shown in Figure 2.5. This graph is similar to to the preceding ones for ORACLE which plot the load rate of the previous 5000 record interval, not the entire load. In this manner, DBMS sensitivities can be more closely observed. For example, the first 51,000 records were loaded at an average of 30.75 records/second but closer examination shows that the first 5,000 records were inserted at a rate close to 40 records/second and a degradation effect occurs to the point where the interval between 45,000 and 50,000 averages about 28 records/second.

Figure 2.5, reveals a choppy pattern of degradation with small peaks and valleys. This oscillation is probably attributed to the hashing formula used to determine the page locations of the records associated with the "CALC" value Profile time. It is suspected that the troughs are associated with a relatively high number of duplicate hits and overflows while the peaks might suggest the "CALC" values have indicated more pages with space available. The overflow condition occurs when the hashing algorithm employed identifies a page to locate a record on that is already full. This happens when the same "data base key" is generated for a number of data values that are hashed. In the FGGE/LIMS application the profile time is an incrementing string of characters that is

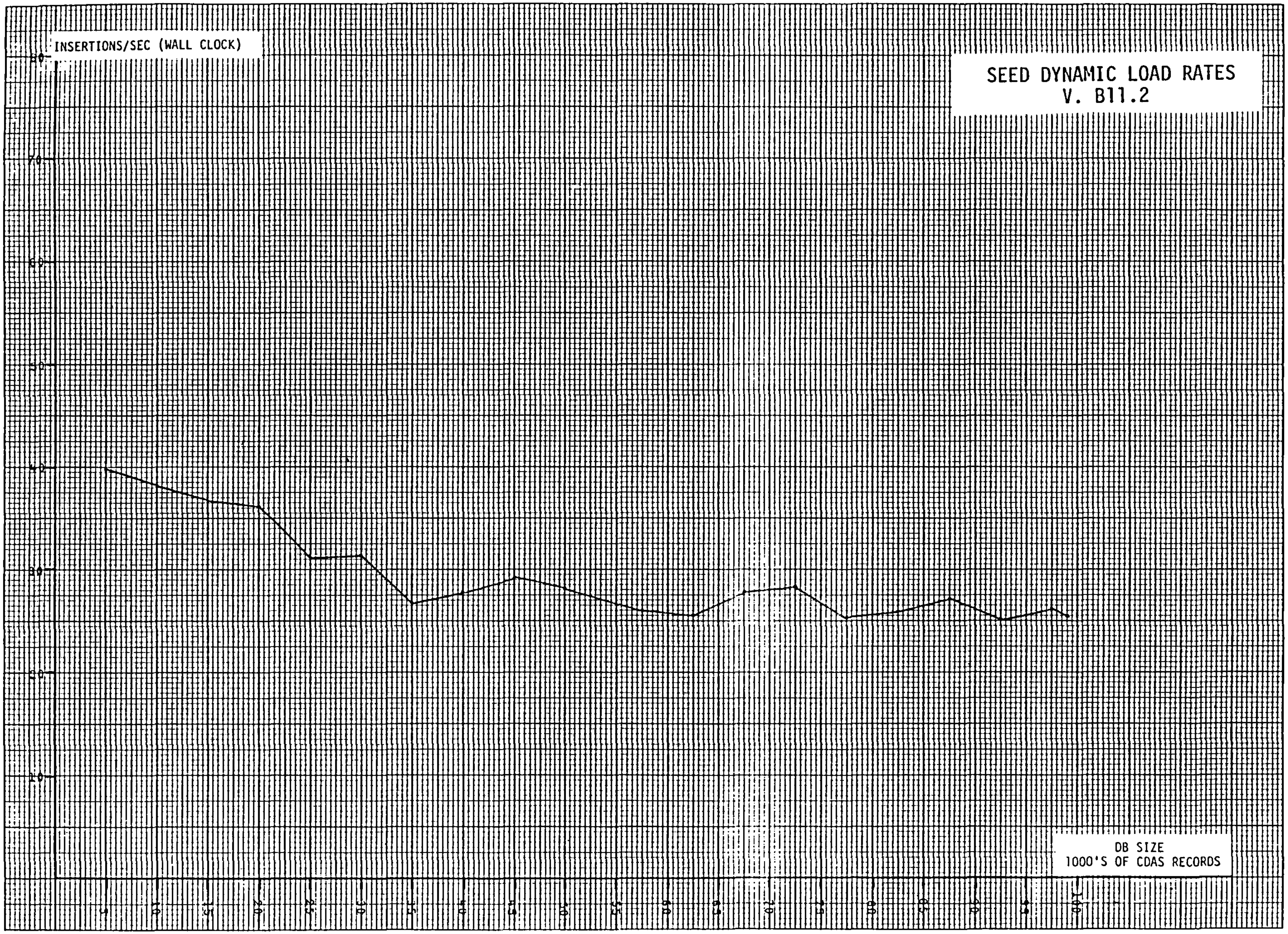

INSERTIONS/SEC (WALL CLOCK)
SEED DYNAMIC LOAD RATES
V. B11.2
80
70
60
50
40
30
20
10
DB SIZE
1000'S OF CDAS RECORDS
5
10
15
20
25
30
35
40
45
50
55
60
65
70
75
80
85
90
95
100

Figure 2.5

very similar in appearance from one value to the next which results in a number of similar data base keys. Also the latitude and longitude values are over a limited range and they too may be causing overflow. Once the overflow occurs, chaining to some other available space must take place burdening SEED with extra overhead to locate an indexed value or a spot for a record. As the profile time changes it periodically results in a data base key which points to a clean page and improves the load rate.

While conducting the load from the 99,000 record level to approximately 180,000, a serious problem arose. After loading approximately 35,000 more records, the load was stopped by the indication of an error in the insertion process. The SEED error returned was:

```
"ERRSTA = 1222 IN STORE"
"RECORD IS NOT CURRENTLY A MEMBER OF SET".
```

Subsequent discussions with IDBS personnel revealed an undefined problem that SEED had encountered in some other applications with large data bases using the dynamic load technique. IDBS personnel believed that the VAX system software was indicating to SEED that more space was available than was actually allocated. This problem was said to be corrected in Version 2.1 of the VAX operating system but, since this version was not available on the benchmark VAX computer at the time, a new data base was defined using the "STATIC" approach. The "STATIC" approach forces SEED to search logically succeeding pages already defined to the data base for enough space to place a record when overflow occurs. This means the data base must be initially defined large enough to hold all the information to be introduced to it or a data base unload (using TROUT) followed by a new "DBINIT" and then a data base reload (using TRIN) would be required to enlarge the data base.

After the decision was made to define the SEED data base using the "STATIC" approach, the data base was rebuilt to the 180,000 record level immediately. It should be recalled when examining the query results that at the first two levels the data base was loaded dynamically while subsequent levels were all associated with the "STATIC" data base specification. The load rates of the first 99,000 records are shown in Figure 2.5. It appears that there is an improvement in performance over the same load range. This may be due to not having to call the operating system to request additional space. Close comparison of Figures 2.5 and 2.6, Part 1 show that the troughs and peaks, although not identical, are very similar in pattern revealing that regardless of whether the data base is "DYNAMIC" or "STATIC", a fluctuation exists.

Figure 2.6, Part 1 also shows the incremental load rates from 100,000 to 187,000 records. It is still apparent that degradation exists but it becomes much more gradual after 130,000 records were in the data base. The load rate has dropped from an initial high of 45 records/second for the first 5,000 records to about 17 records/second to load 5,000 records at the 180,000 point. Figure 2-6, Parts 1 and 2, show the load rates for the interval between 187,000 and 434,000 records. The degradation seen over the 100,000 to 180,000 range is continued over the range from 180,000 to about 265,000 records. At this point a new tendency was observed. Performance improved but a roller-coaster effect is noted on the graph with a great deal of fluctuation between intervals. The increased performance was suspected beforehand and is probably caused by the profile time finally transcending from 1978 to 1979. If one recalls, profile time is a ten-character string whose first four characters consist of year and month. Up to the 270,000 mark all profile times had the same first four characters, 7812 (year 1978, month 12). Around the 270,000 mark the same four characters switch to 7901. Improved performance was anticipated because it was thought the hashing algorithm would be more likely to arrive at new pages that would not result in overflow (at least temporarily).

The improvement in performance is obvious from looking at Figure 2-6, Part 2, even if it is only a four or five record/second improvement for the interval between 270,000 and 330,000 records. More obvious to the eye is the amplitude between successive observations. The choppiness is so marked (even though the results are averaged over 5000 insertions) that the only explaination supporting the behavior is the overflowing overhead of duplicate data base keys. Since the data base is now "Static" the overflows are going on successive pages with space available. Perhaps the overhead of searching for overflow space periodically rises and falls like a sine wave due to the nature of the inserted values. There is no other reason related to either the load software or the VAX system which would contribute to this behavior other than the chance that disk seek times might periodically add some small overhead. (This in a worse case would only add 56 milleseconds to a disk read or write using the RP06 disk drives.) The load rates toward the end of this load indicated degradation to around 12 records/second.

The next load was performed in three parts and advanced the data base to about 1,039,000 records. The load rates are displayed in Figure 2-6,Parts 3, 4 and 5. The version of SEED used for these loads was updated to B11.3. An inspection of Figure 2-6 reveals periods of relative stability with slight degradation each followed by a period of significant oscillation at an observably higher rate of performance. As before, this is attributed to the hashing algorithm and the profile time values. A general degradation has persisted throughout the load to a point where the last 50,000 records have averaged about 4.5 records/second for their insertion rate.

An examination of Figure 2.6, Part 5, shows the omission of load results from 830,000 to 865,00. This omission is due to a VAX system "crash" which occurred in the midst of a SEED load. No other users were signed on or active when the "crash" occurred but there is no evidence to indicate the SEED system was in any way responsible for the system failure. Further, it was determined that the data base remained in a useable state. After deleting the last partial set of entry records and the associated profile record the load process was restarted without problem and completed normally.

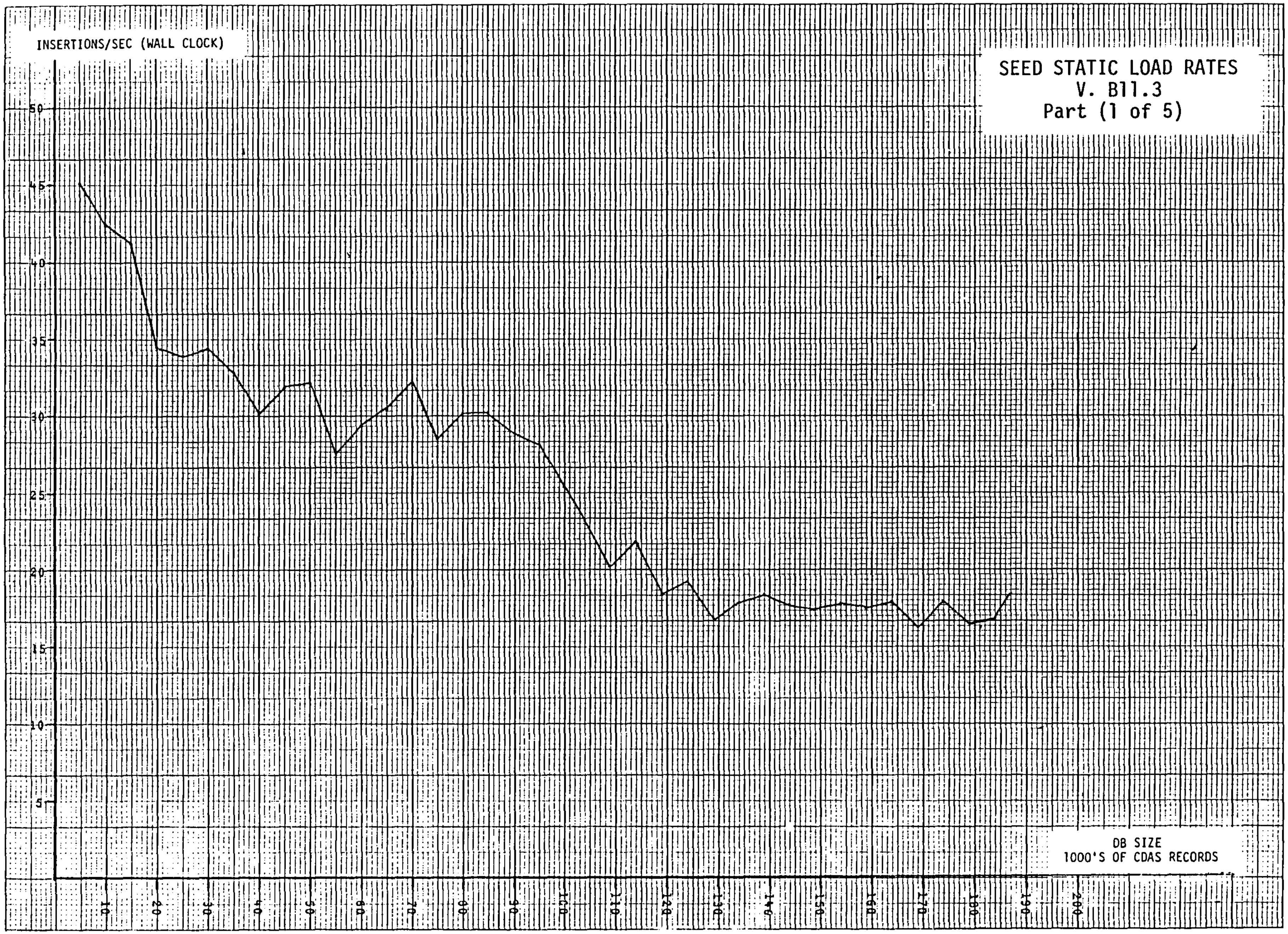


Figure 2.6, Part 1

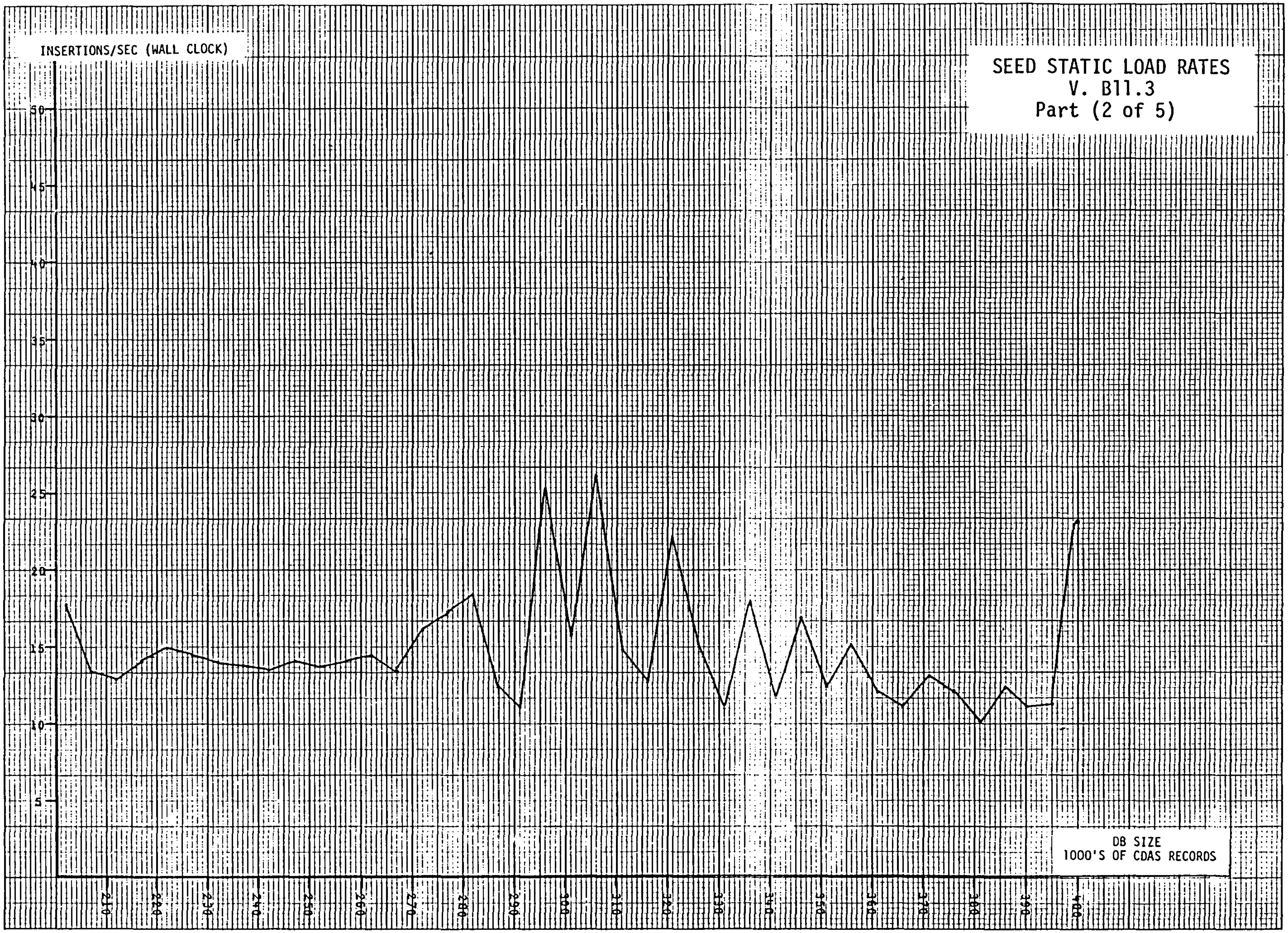


Figure 2.6, Part 2

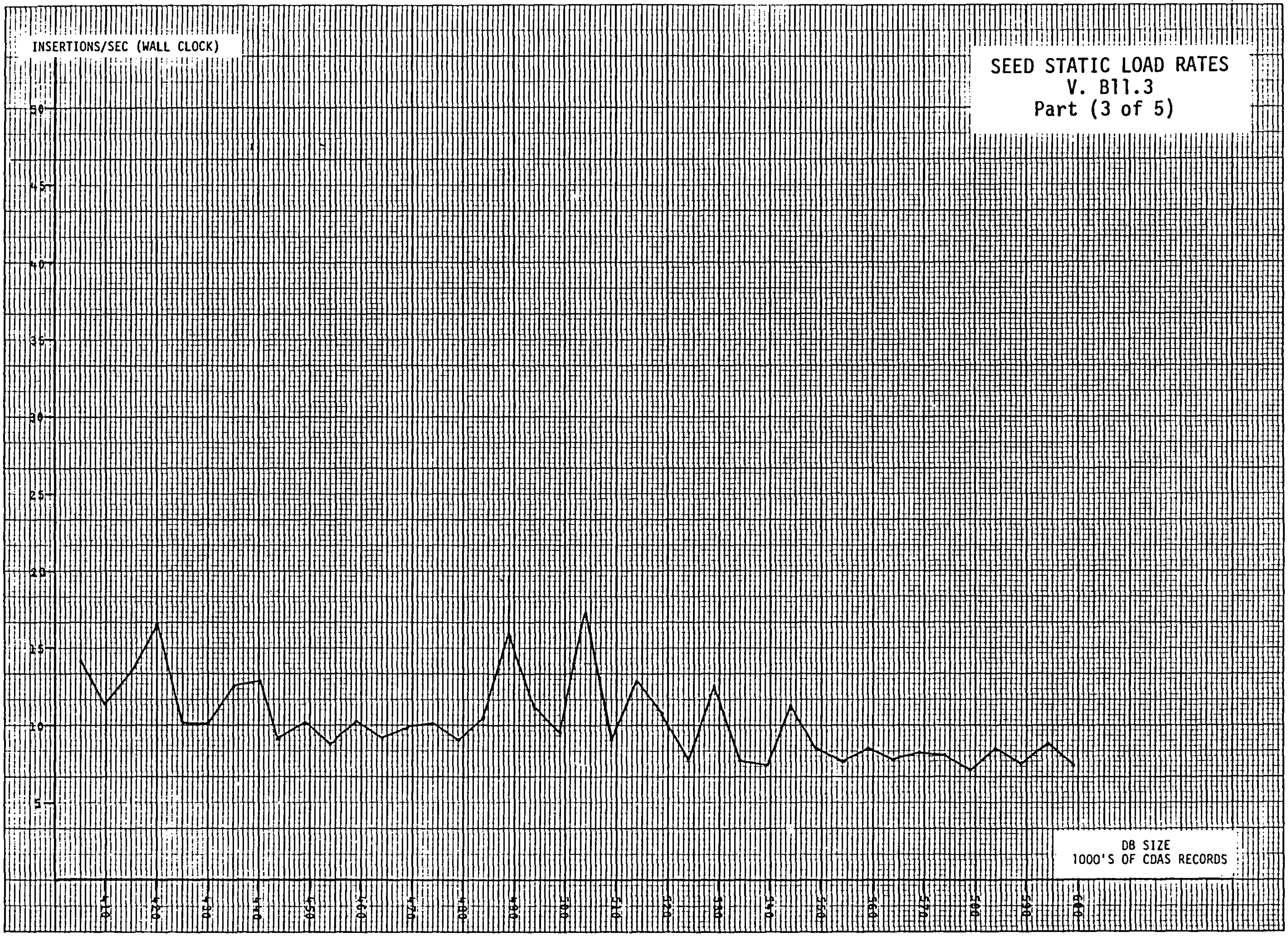


Figure 2.6, Part 3

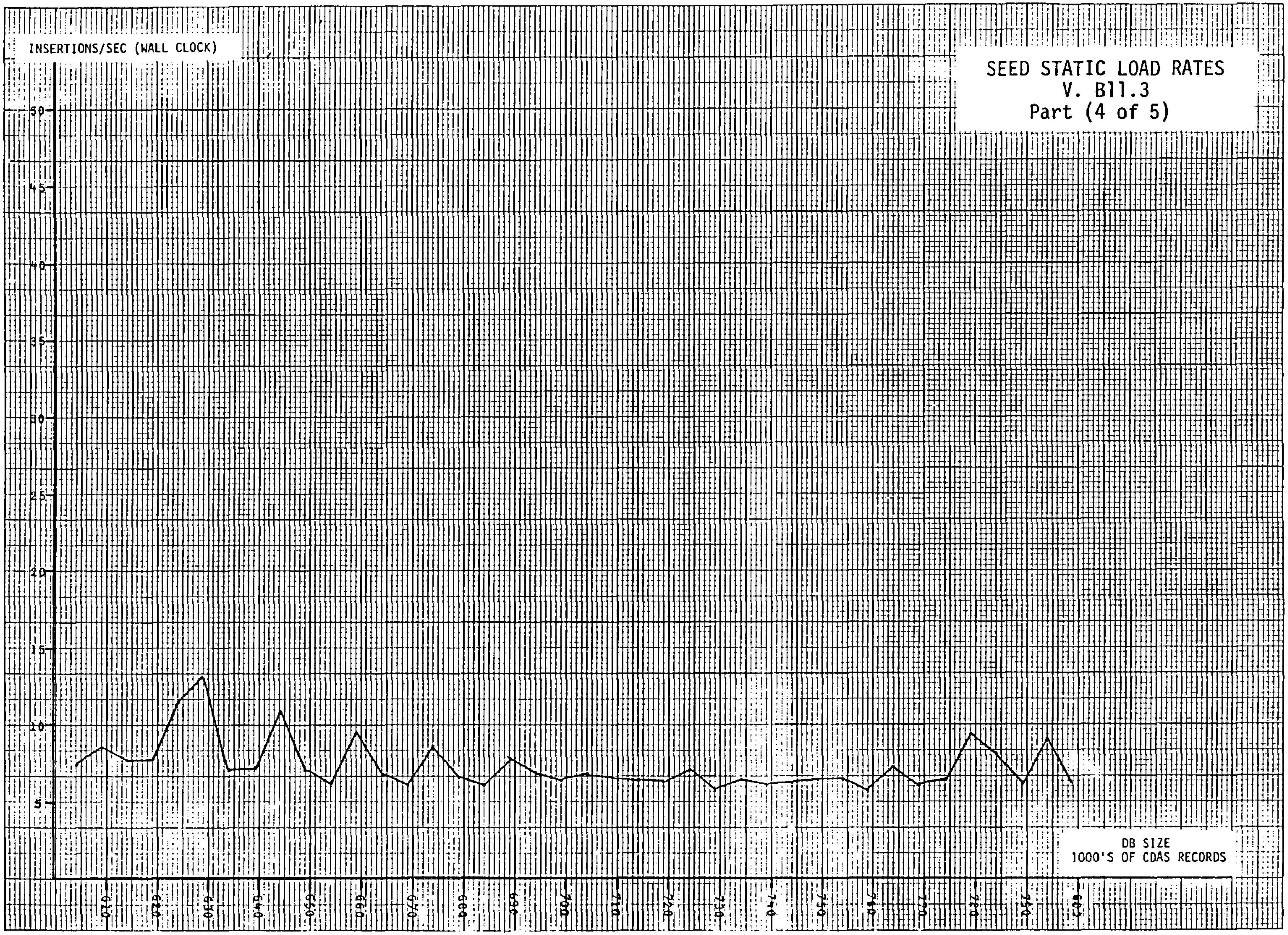


Figure 2.6, Part 4

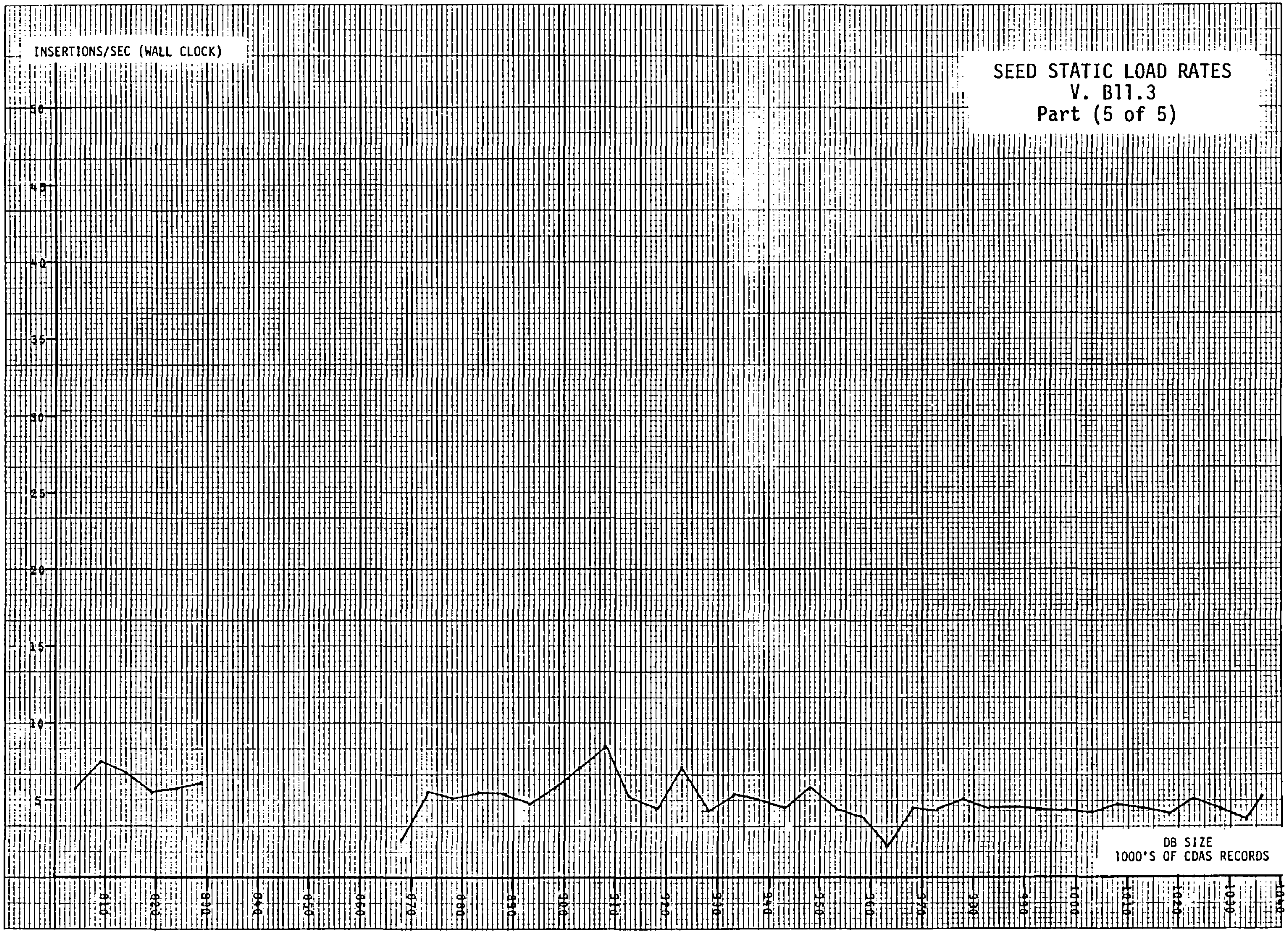


Figure 2.6, Part 5

Use of DBSTAT (SEED's Data Base Statistics Package) revealed that the data base consumed 19,746,622 bytes. This represents 48.9 percent of the figure determined in Section 2.1.2.2.2. The data base contains about 48 percent of the data originally planned for indicating that the derived figure was very close to the actual consumption rates. An overall summary of the load process from 0 to 1,039,000 records is included in Figure 2.7. It has been smoothed somewhat, but provides a broader view of the load rates. Another observation about the loading process was that in an uncontested environment the ratio of CPU time consumed by SEED processing to elapsed wall clock time was consistently around 48% for all loads.

Subsequent to the loads discussed above, several more loads were conducted to experiment with the DIRECT loading capability available in SEED. The approach used defined separate areas (or files) for the Profiles and Entries. The Entry records' "location mode" was defined as DIRECT and the entries were inserted in a sequential order controlled by the currency pointer in that area. This plan would mean that more profiles could be stored on a data base page (since no Entry data is present) thus reducing the likelihood of overflows caused by duplicate hash code values. Fewer overflows would mean faster loading. Theoretically, this approach would increase response time when querying the entry data via the Profile time since Profile and Entry data would be in separate areas instead of closely packed on the same physical page. Figure 2.8 shows the results of loading the first 200,000 records of the FGGE/LIMS data using the direct technique. There still exists some degradation and choppiness in the load rates but comparison with the rates previously attained (see Figure 2.6) reveals a significant improvement in load performance. The last 50,000 records appear to have load rates that are almost twice as fast using the DIRECT approach.

The results obtained in this experiment are not conclusive because query results are not available to compare access rates but they do point out several things. The improvement in performance is so significant

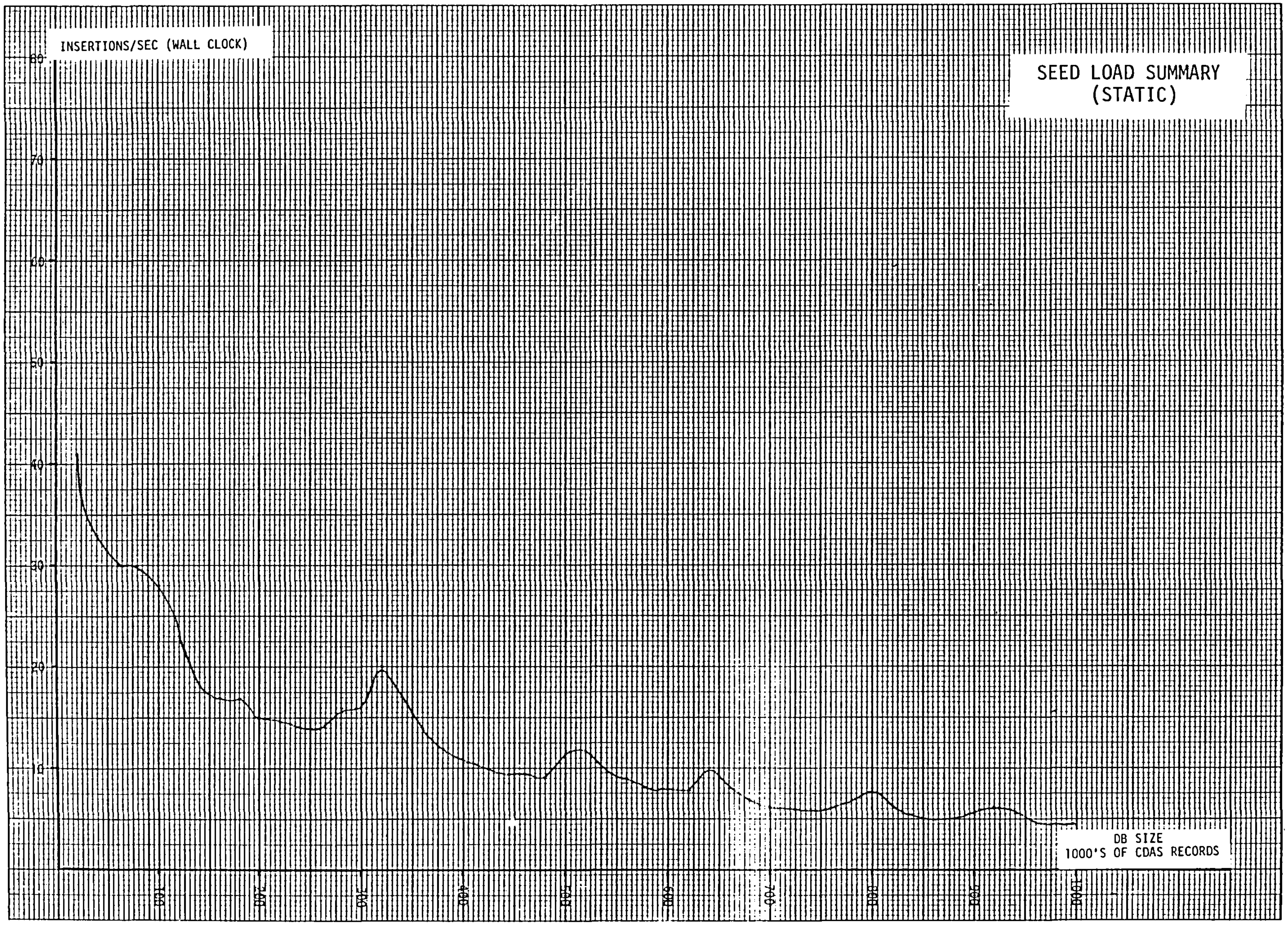

INSERTIONS/SEC (WALL CLOCK)
80
70
60
50
40
30
20
10
SEED LOAD SUMMARY (STATIC)
DB SIZE
1000'S OF CDAS RECORDS
100
200
300
400
500
600
700
800
900
1000

Figure 2.7

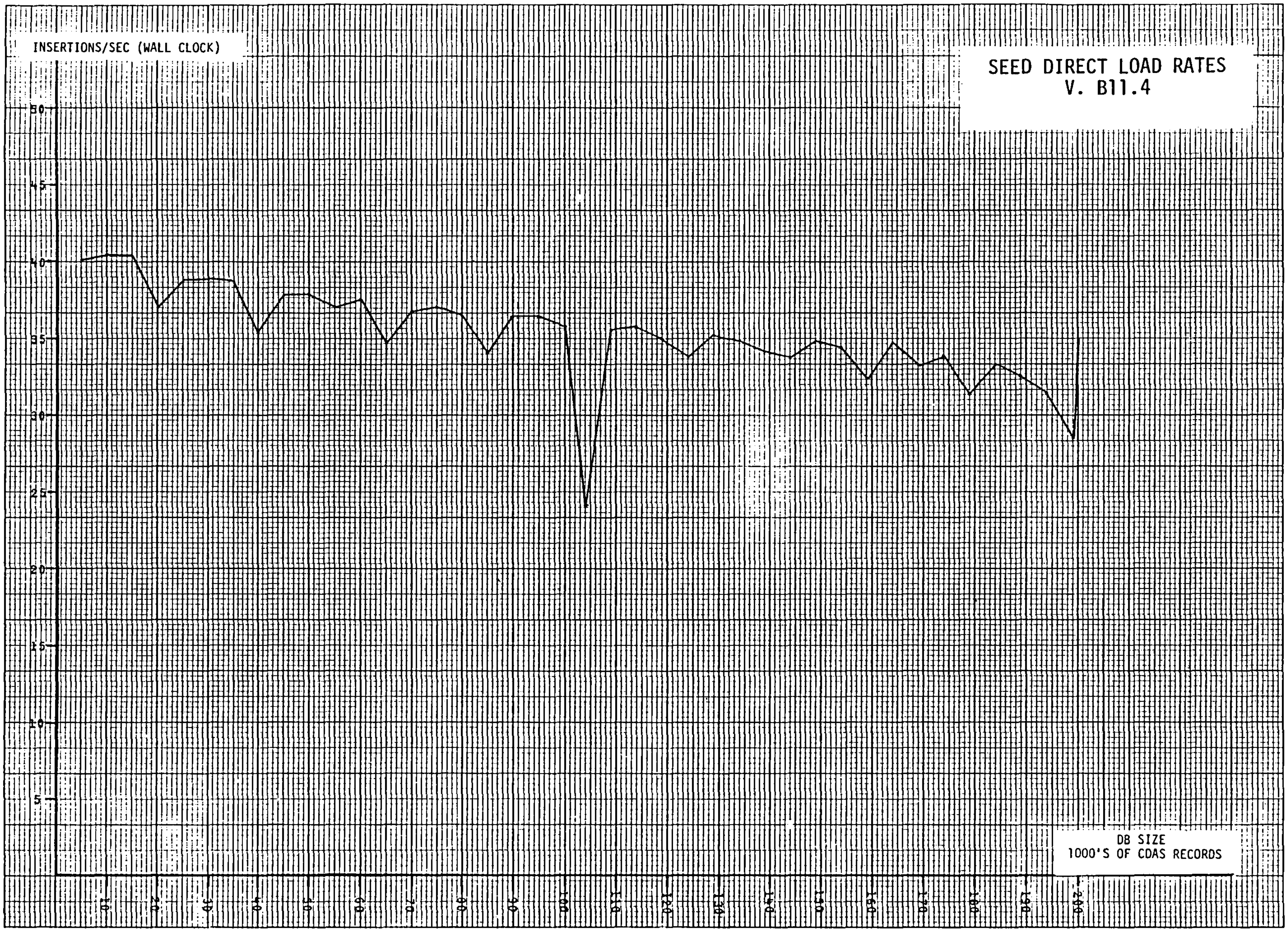

INSERTIONS/SEC (WALL CLOCK)
SEED DIRECT LOAD RATES
V. B11.4
50
45
40
35
30
25
20
15
10
5
DB SIZE
1000'S OF CDAS RECORDS
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
170
180
190
200

Figure 2.8

that the DIRECT approach should be considered when load rates are critical. The theoretical advantage in query response time of having Profile time and Entry data maintained in physically adjacent space may be minimized somewhat if that approach is subject to a significant amount of page overflowing, which creates more disk I/O , thus nullifying the original gain. Underlying all of this is the matter of hashing. Given a particular type of data, one might well be able to produce a hashing algorithm which produces a random distribution of hash codes and therefore reduces duplicate hits greatly. The benefit of such an algorithm for a particular application could be enormous if it is definable. Theoretically, production of an algorithm that generated random values for the data being used would result in load rates that were essentially uniform and would have no degradation. But production of such an algorithm may not be a trivial problem for a given set of data.

2.2.4 RIM Load Results

As was previously stated, the RIM DBMS was introduced to the benchmarking procedure just prior to the beginning of the tests. As a result, a certain amount of trial and error was required during initial loads that would ordinarily have taken place in a preliminary checkout. The initial RIM data base design was based on the ORACLE design since both were relational systems and shared the tabular concept of data representation. This approach would also permit direct comparison of benchmark test results.

Figure 2.9, Part 1, shows the 5000 record interval rates for the initial 50,000 records inserted into the data base. It was easy to see that a significant amount of degradation occurred within the first 15,000 records. In fact, the asymptotic nature of the graph was alarming because it was apparent that building a large data base would require more time than could reasonably be dedicated for it. As a result, a discussion of the data base design and this initial load was held with Mr.

Wayne J. Erickson who was instrumental in developing the RIM system for the Boeing Company. His explanation for the load rates involved the profile time field and the associated inverted file or B-Tree used for indexing it.

One may recall that the profile time field was stored as a ten-character string and was specified as an indexed field. Since the first four characters represent year and month, all of the values of these four characters are the same in the initial load. RIM builds its inverted file for an indexed field based only on the initial four bytes of the field which, in this case, corresponds to the year and month specification of profile time. All keys whose initial four bytes match are chained together as duplicates. Further complicating the issue, duplicates are added at the end of the chain (rather than the beginning) which required RIM to transcend the entire chain to insert each new profile time value. (Mr. Erickson explained that a future version of RIM would maintain a pointer to the end of the chain eliminating the need to read through it when adding a duplicate.)

It was then decided that the second load, from 50,000 to almost 100,000 records, would only contain indexed values for the profile count values repeated in both the PROFILE and ENTRY tables. The profile time, latitude, and longitude fields in the PROFILE table would not be indexed during the load. For this reason, direct comparison with ORACLE load rates over this range of the loading procedure must recall that ORACLE is building an additional index that RIM is not. Figure 2.9, Part 1, depicts the results of this load and demonstrates a tremendous improvement in load performance. Similar to ORACLE's load software, the load software for RIM must obtain the profile count value to begin the load with. The RIM load software does this by sequentially reading to the end of the Profile Table to acquire the last profile count value used. This accounts for the initial increase in load rate between 57,000 and 62,000 records. The rapid decline in performance from 67,000 to 82,000 records is somewhat a mystery in light of the "leveling off" between 82,000 and

100,000 records. In any case, the minimization of indexing produced a dramatic improvement in load performance, increasing load rates by about a factor of 50 between the original load at the 45,000 to 50,000 record level and the new load at 50,000 to 60,000.

Following the second load the latitude and longitude fields of the PROFILE table were indexed via the RIM "BUILD KEY" command so that queries could be tested. This required about 14 minutes of wall clock time and almost five minutes of CPU time to accomplish for the PROFILE table which now contained approximately 5,900 rows. It was now confirmed that the profile time was the primary cause of the original degradation (as Mr. Erickson had indicated) and the third load (from 99,000 to 180,000) omitted only the profile time index from the original design. Figure 2.9, Part 1, shows the load rates from 100,000 to 180,000 for RIM. Over the course of this load a general degradation can be observed from about 22 records/second down to about 18 records/second. There are small irregularities which can be observed in Figure 2.9, Part 1, indicating that unknown overheads and efficiencies exist periodically, but what they might be is undetermined. Possibly the dynamic allocation of disk space as the data base grows contributes to periodic irregularities, and the construction of inverted files could be the cause of some fluctuations in load performance.

The next RIM load took the data base from about 180,000 up to 437,000 records. This load was especially interesting because it was one of the few known cases where contention with other VAX users occurred. The load was begun at approximately 6:00 a.m. and the VAX computer was reserved for stand-alone use until 9:00 a.m., but the load was estimated to require four hours without contention and would inevitably take longer when contention was introduced by other users. The load began with an insertion rate of about 20 records/second at the 200,000 to 210,000 record level and gradually decreased with irregularities similar to the previous load. At the 360,000 record level the insertion rate had degraded to about 13.5 records/second. At about this point user contention

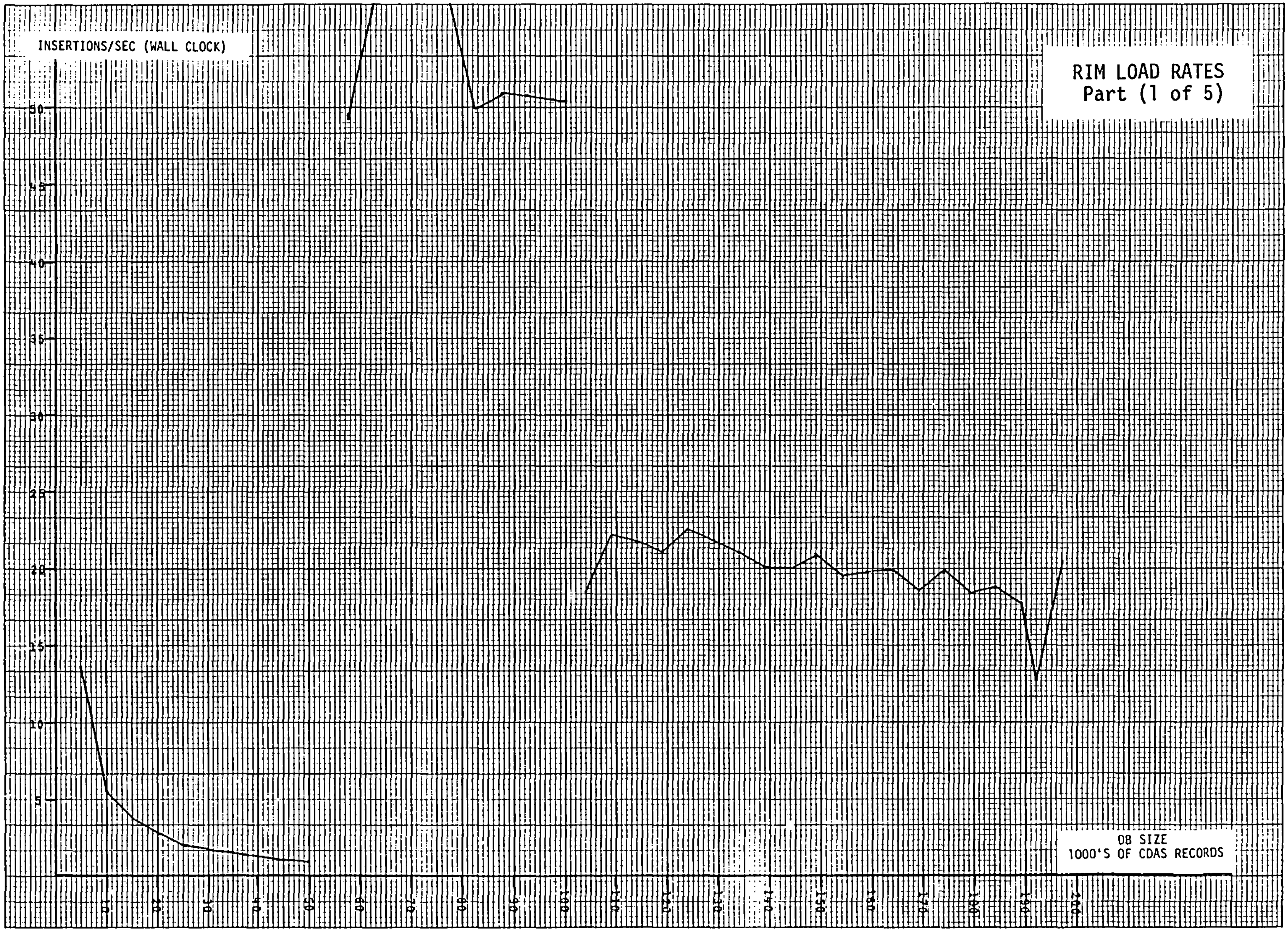


Figure 2.9, Part 1

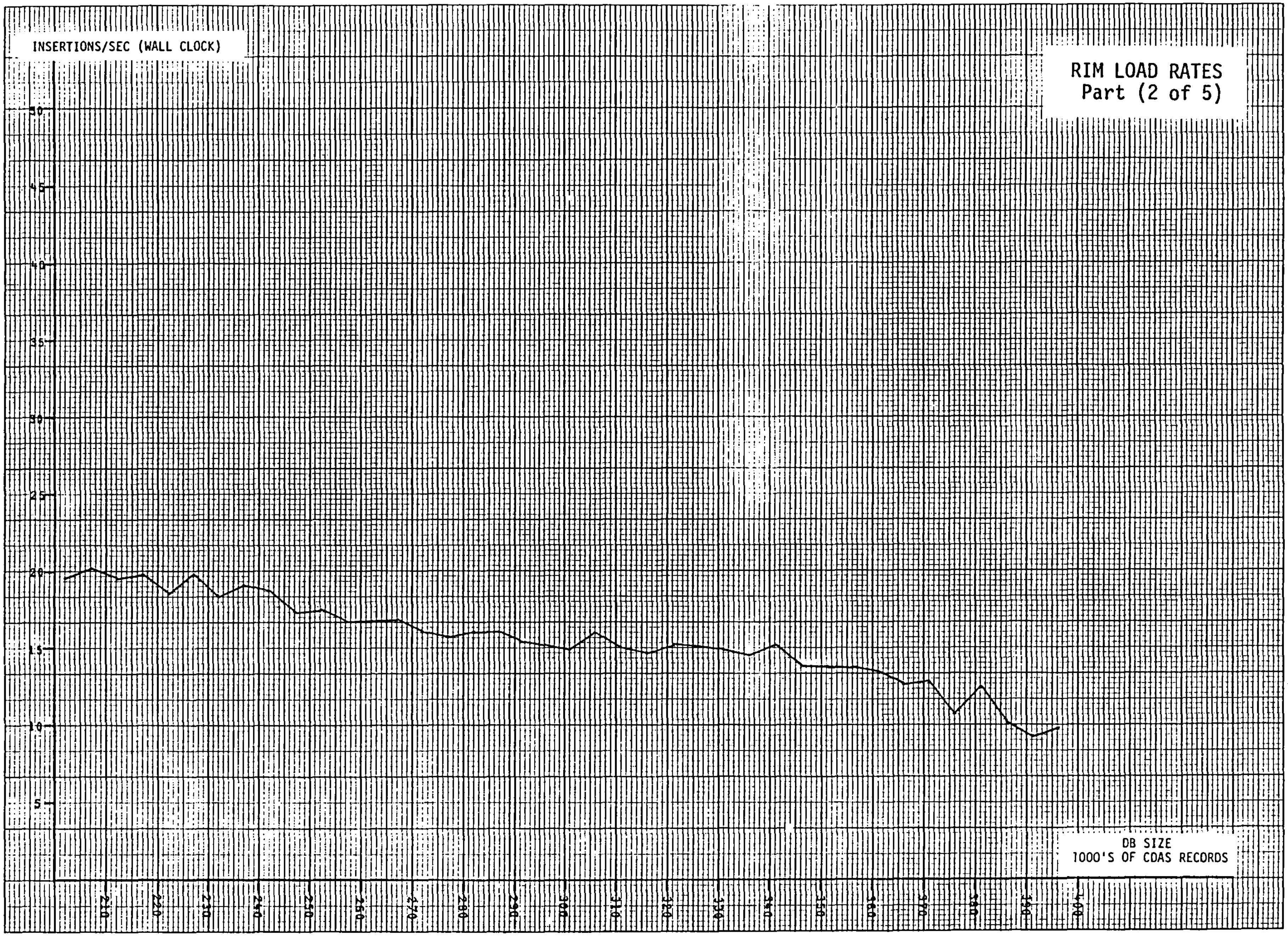


Figure 2.9, Part 2

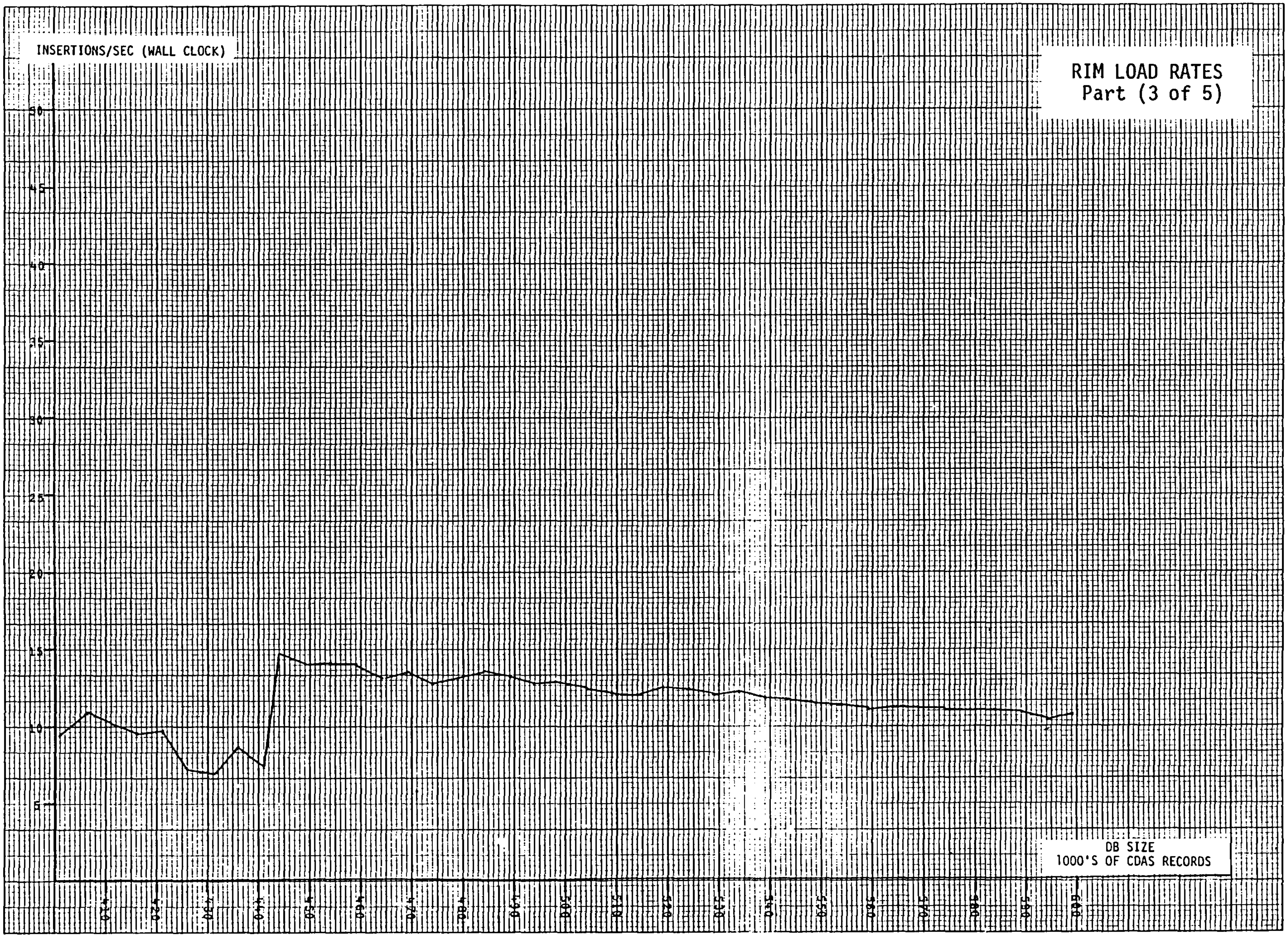


Figure 2.9, Part 3

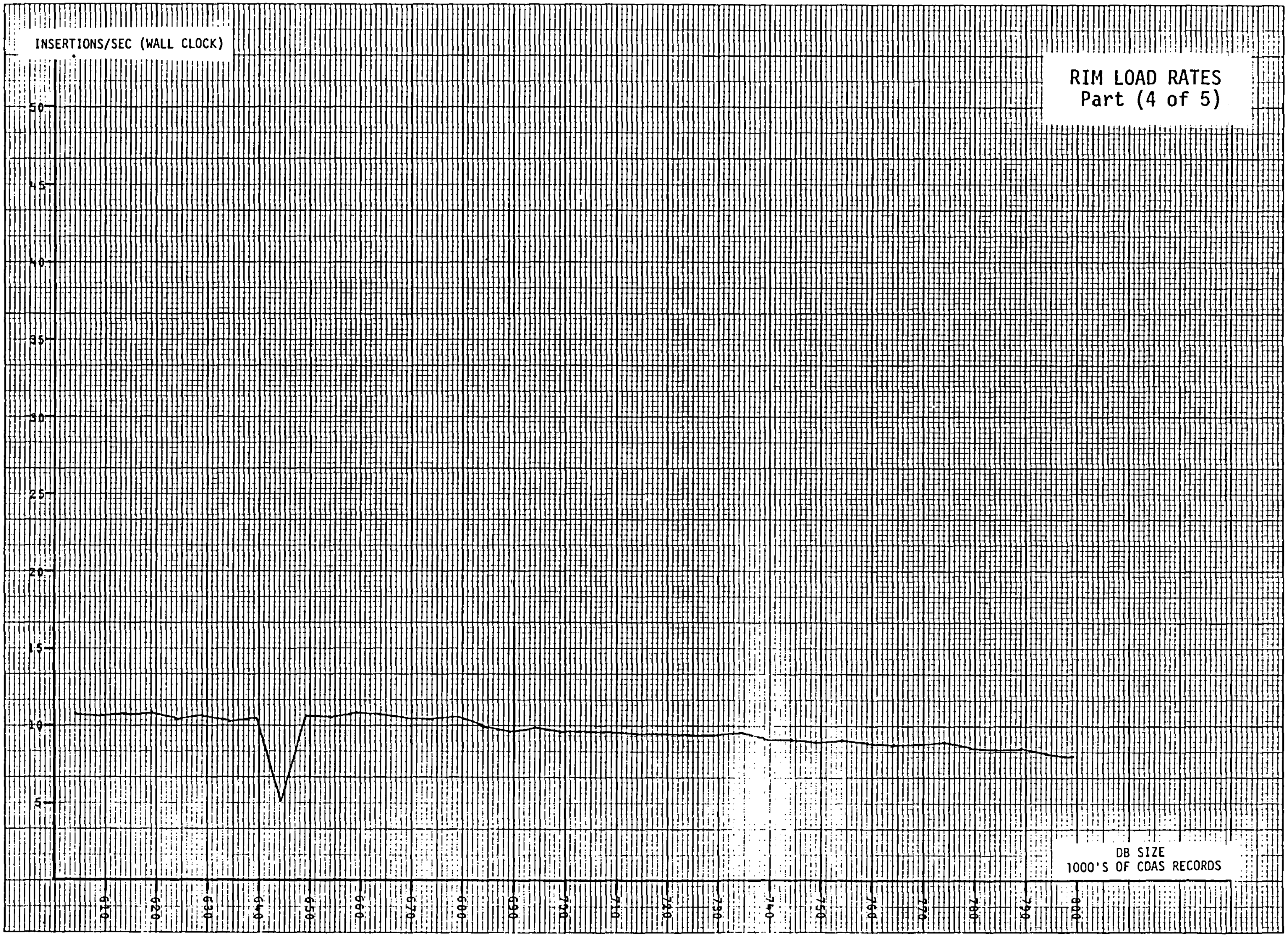


Figure 2.9, Part 4

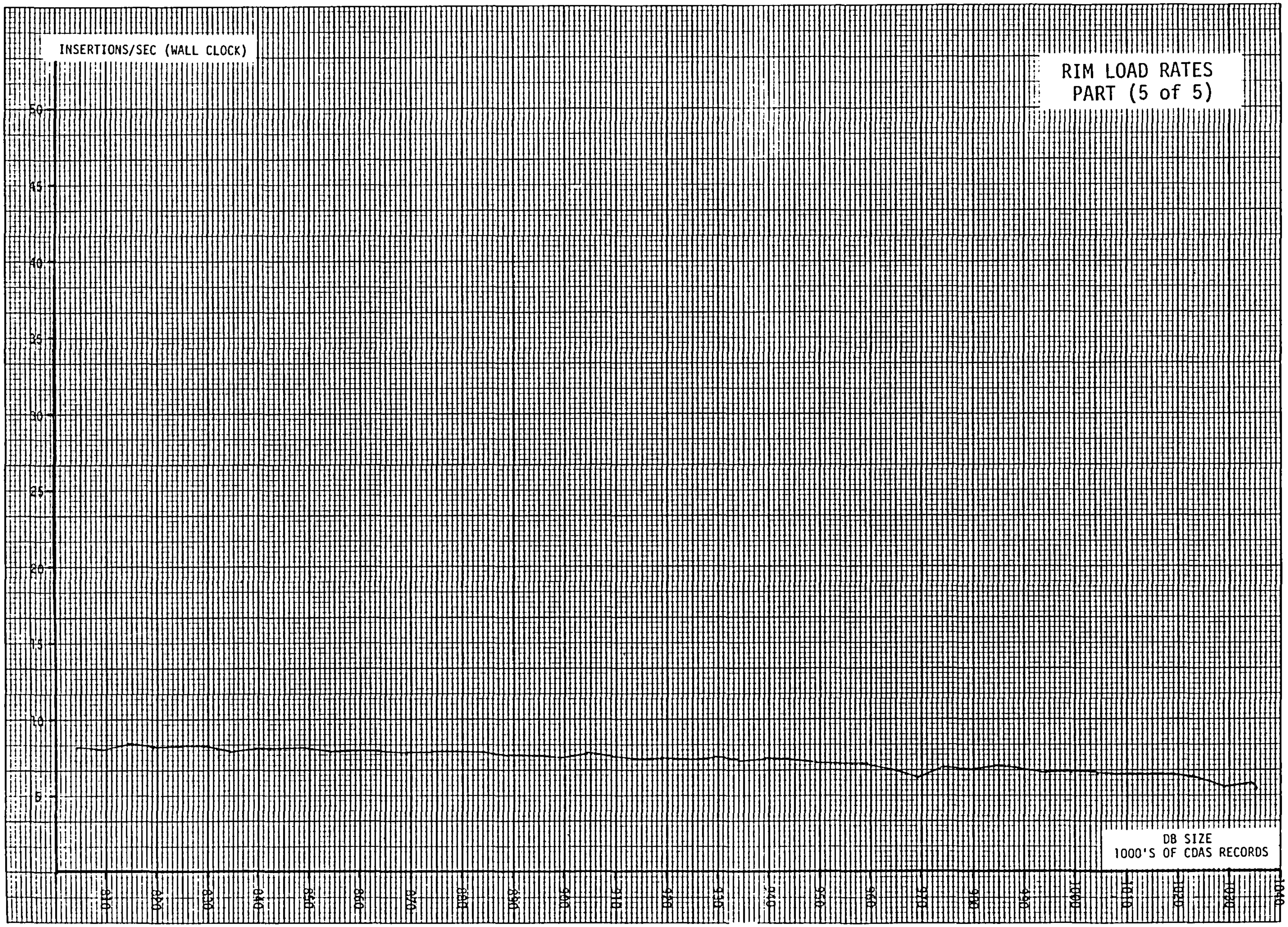
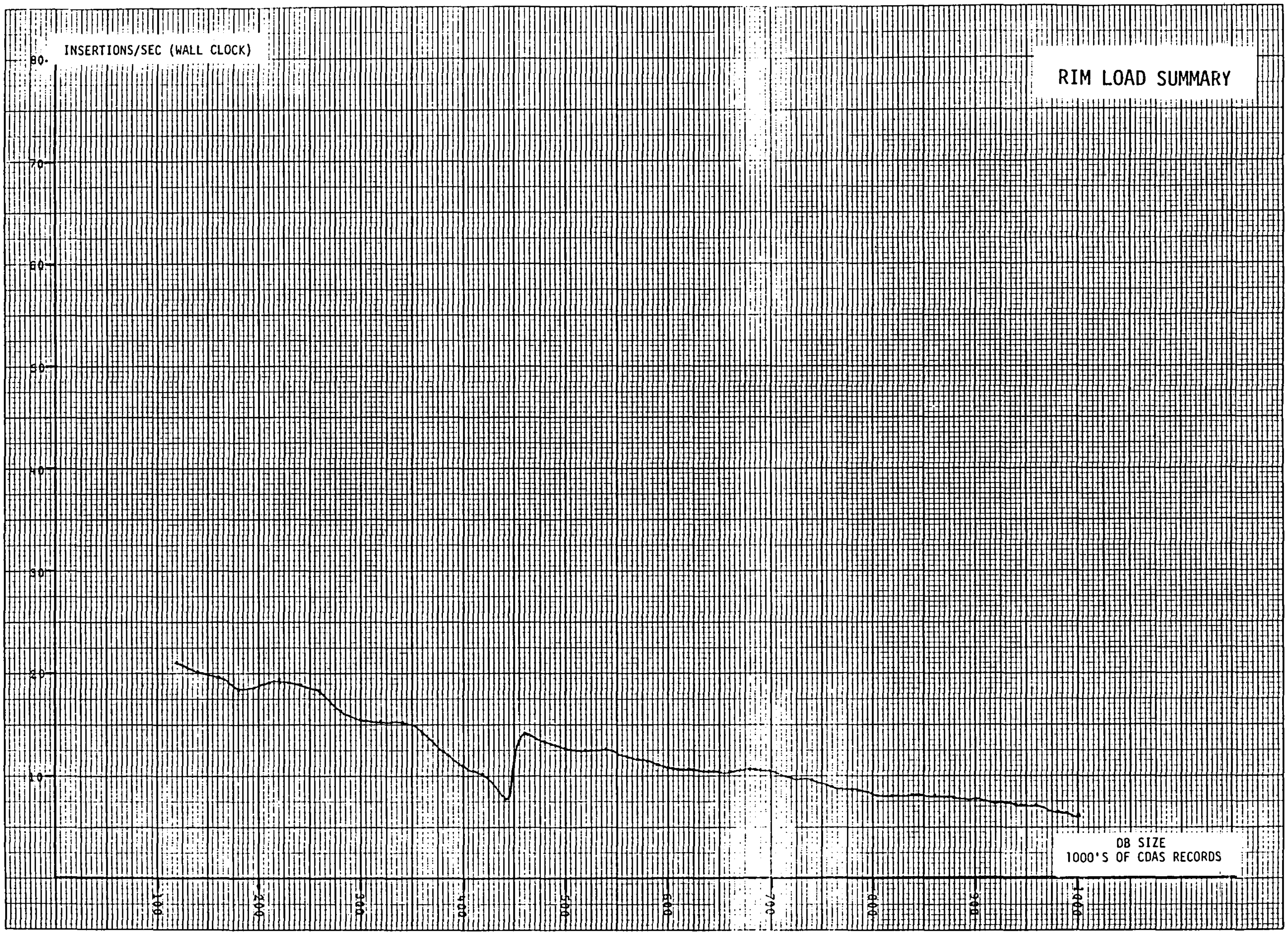


Figure 2.9, Part 5

began. A significant change in the degradation rate can be observed over the range from 370,000 to 440,000 records. The irregularities became more pronounced and the load rate was eventually down to about 7.5 records/second. The effect of contention was verified when the next load began without contention at more than 14 records/second and did not drop to the 7.5 rate again (without contention) until the data base reached the 900,000 record level.

The load required to reach the 1,035,000 record level was conducted in two parts and is shown in Figure 2.9, Parts 3, 4 and 5. Examination of these graphs shows a gradual degradation with decreasing irregularities. In fact, the rates are so uniform that accurate predictions of load rates for subsequent loads could easily be made. The final three rates observed after 1,020,000 records were also exposed to user contention which explains their increased degradation. Prior to that point the load rate had degraded to approximately 6.5 records/second.

Figure 2.10 summarizes the load rate figures. Rates for the initial 100,000 records are not included since they were loaded with different indexing requirements than the remainder of the load. The contention at the 380,000 to 440,000 discussed above stands out dramatically on this graph. The final RIM data base contains three files totaling 40,366,080 bytes. RIM apparently uses one file to contain data dictionary type information, a second to contain the tabular records, and a third for the inverted files used for indexing key values. For the LIMS application the data dictionary information used .1% of the data base space, the tabular data used 71.9%, and the inverted files required 28% of the space. An examination of the CPU time consumed during loads versus the wall clock time shows a decrease in the proportion of CPU time consumed per second of wall clock time as the data base enlarges. The 100,000 to 189,000 load used approximately 34% of each available second while the 189,000 to 439,000 step used about 27%. This 27% figure is influenced by the contention experienced between 370,000 and 439,000 as discussed



INSERTIONS/SEC (WALL CLOCK)
RIM LOAD SUMMARY
80
70
60
50
40
30
20
10
100
200
300
400
500
600
700
800
900
1000
DB SIZE
1000'S OF CDAS RECORDS

Figure 2.10

above. Up to the 370,000 level the percentage was about 30%. Between 439,000 and 639,000 records the CPU use was about 29% and between 639,000 and 1,039,000 records the percent of use was down to about 24%. This reduction appears to be due to an increase in disk I/O as the data base gets bigger which could be expected. Any comparison of load related figures directly with ORACLE's must take into account the omission of profile time from the inverted files. This omission results in less space consumed in the inverted files and reduces the effort required to load the same data loaded into the ORACLE and SEED applications. Without the omission, testing could not have been completed for the data base sizes chosen due to RIM's inefficiency in indexing the Profile time value.

2.3 Query Testing Of The Data Bases

2.3.1 Test Plan

The test plan had to include demonstrations of analogous functions in all three DBMS packages. A particular test had to be designed so that results could be compared. Since all the systems were managing the same data and since the designs chosen for the data bases were conceptually similar (see Section 2.1.2), tests could be constructed that would facilitate the desired comparisons. Because of the number of times the query tests were to be repeated, the urgency associated with the completion of the testing and the availability of "stand alone" VAX computer time the tests had to be relatively concise.

The tests that evolved were both simple and straight forward but would measure the primary capabilities required of the packages. Similar to the load measurements, these tests measured elapsed wall clock time, CPU time, direct I/O's and page faults. Direct I/O's are related to the number of actual read and write operations made by the monitored software but would not include the number of I/O's which may result due to operating system support of the active task such as reading in software when a virtual address is determined not to reside in main memory. Page faults are a measure of the number of times the monitored software addresses an instruction or data cell directly that is not presently in main memory resulting in a disk read to retrieve it. They were also conducted in a "stand-alone" fashion so that external interference would be eliminated. This means that the results are best-case numbers that might not be obtainable when other user contention existed. The tests were normally performed using the provided query languages and repeated using FORTRAN routines exercising the appropriate Host Language Interface (HLI) to accomplish the same function or operation.

Table 2-8, Benchmarking Operations, summarizes the functions performed. Items 1 and 3 should demonstrate the effectiveness of the indexing techniques implemented by each DBMS package. Items 2 and 4 should demonstrate the overhead or cost associated with accessing all the Profile Information (Records in SEED and Rows in ORACLE and RIM) and all the Entry data respectively by searching on non-indexed fields. The fifth consists of three separate but similar queries that demonstrate the cost of compound selection criterion for indexed and non-indexed fields. Item 6 was only performed using the 52,000 record data base. Its purpose was to determine the cost of using the available DBMS sort capabilities. Items 7 and 8 measure the incremental cost to add and delete a record (or row) to and from the data base. Item 9 was included as a measure of control so that this overhead could be accounted for in some of the preceding operations conducted using a Terminal Interface (TI) language. This was necessary when determining the net cost of the operations using the provided query language. All of the TI tests were implemented via VAX command files to eliminate the variability of human typing rates. A side effect of this approach is that the measurements yielded in the TI tests include the overhead of opening and closing the data base. The inclusion of Item 9 was done to provide a basis for determining a net figure by presenting the cost of simply opening and closing the data base. One can derive a net figure by subtracting the Item 9 results from the other query results.

TABLE 2-8

BENCHMARK OPERATIONS

1. Using an indexed field, locate a specific Profile related value.

2. Sequentially access all the Profile records.

3. Using an indexed field, locate a specific Entry related value.

4. Sequentially access all the Entry records.

5. Establish the cost of compound selection criteria
   - A.) With a single indexed value as selection criteria (control)
   - B.) With two indexed values as selection criteria
   - C.) With an indexed value and a non-indexed value as selction criteria

6. Measure sort capability
   - A.) Unsorted (control)
   - B.) Sorted

7. Incremental addition and deletion of a Profile value
   - A.) Insertion
   - B.) Deletion

8. Incremental addition and deletion of an Entry value
   - A.) Insertion
   - B.) Deletion

9. Open and close data base without intermediate operations

The test functions described in Table 2-8 are consistent with the goals of the benchmark and remained within the boundaries of the constraints imposed on the testing. Although the tests do not comprehensively measure all aspects of data base performance, they do provide a basis for evaluating the fundamental response rates and resource demands for the primary DBMS functions. Because of this, the results can be used to compare a single DBMS's sensitivity to data base size from 50,000 records to a million records or to compare its interactive query responses to a host language interface performing similar functions or to compare one DBMS's performance to another's in the same frame of reference.

2.3.2 ORACLE Test Results

As noted in the ORACLE load discussion (Section 2.2.2), a new version of ORACLE was provided after the 189,000 row level had been reached. Upon receiving the new version, which was a test release version 2.3, tests were repeated at the 52,000 row level and then performed at the 439,000 level. The results at the 52,000, 439,000, 1,039,000 row levels were produced with an early 2.3 version while the results at the 99,000 and 189,000 row levels were derived using version 2.2. Comparison of the 52,000 row results done with version 2.3 with that originally done with version 2.2 showed an improvement in performance. This must be considered when looking at the 99,000 and 189,000 results. The results of the tests are discussed in the following paragraphs and the actual results appear in Appendix I.

As mentioned earlier ORACLE presents a problem in measuring resource utilization because a detached process is initiated when a user logs onto the data base and performs a majority of the actual data base processing instead of the user's task. By examining the System Activity Log, figures were determined for the detached process for all the version 2.3 TI tests. These figures are identified in Appendix I in the column labled ORAAAA (which was the name of the created detached process). The

ORAAAA figures provided at the 99,000 and 189,000 data base levels are not totally accurate. They did not originate from the System Activity Log but, instead, were derived from the interactive system monitor. Since the figures for the detached process disappear on the interactive display when the process terminates the figures provided for the 99,000 and 189,000 are the last observed and as such are minimum utilization and quite likely are less than the actual figures.

As noted in the previous section the TI figures include overhead for opening and closing the data base and this is true for the detached process figures as well. Again one might derive an estimated net figure for a particular TI test by subtracting the Item 9 ORAAAA results from the ORAAAA results of interest. Because the HLI tests were conducted as a single process the System Activity Log could not be utilized to determine detached process figures for each test. However, the net figures derivable in the manner discussed above for the TI tests are reasonable measures of what one might approximate for the detached process of an HLI test.

The number 1 Query in the test plan for ORACLE was:

```
SELECT P_TIME FROM PROFILE WHERE P_TIME = 'YYMMDDHHMM'.
```

The actual time value used was the largest Profile time available in the data base at the time of the particular test. Examination of the results of this query for both the Host Language Inerface (HLI) and the Terminal Interface (TI) shows that the indexing algorithms used by ORACLE are effective, since the access times are consistently similar and reveal no significant degradation as a result of increased data base size.

The number 3 query in the test plan for ORACLE was:

```
SELECT * FROM ENTRY WHERE PROFILE_CNT# = NNNNN.
```

The domain "PROFILE_CNT# was the only indexed field in the ENTRY table. This particular query measures essentially the same capabilities as number 1 but for a much larger range in table size. The number of rows in the PROFILE table varied from about 3,000 to about 60,000 while the ENTRY table varied from about 50,000 to about 980,000 rows. The results were consistently similar for all five levels of data base size. Comparison with the number 1 results showed that the number 3 results were on the order of twice as large. The cause of this discrepancy might be attributed to the increased data base size but that does not appear to be the case since the largest table size in number 1 was greater than the smallest table size in number 3 and the 2 to 1 ratio still exists. The most likely cause for this discrepancy is the fact that normally 16 rows were supplied in answer to query 3. This additional I/O is probably the cause of the increase in response overhead. Number 1 used a 10 character field as an index while number 3 used an integer. The results do not indicate a significant cost in terms of response time or I/O's for managing the character field verses the integer field. (This conclusion is based on a limited amount of empirical evidence and the nature of the values of the strings might have allowed ORACLE's key compression to be applied with a great deal of effectiveness which may bias the evidence).

Query number 2 and query number 4 are related in the same manner as number 1 and 3 were. The function was to measure the time required to pass through all the information in the respective PROFILE and ENTRY table. The results should aid in gaining an appreciation of the overhead required to make queries based on non-indexed domains or to access all rows in a table. The query in number 2 was:

```
SELECT P_TIME FROM PROFILE WHERE TAPE_ID = "XXXXXX".
```

The value "XXXXXX" was actually used in this instance so that no matches would be found. The query in number 4 was:

```
SELECT COUNT (*) FROM ENTRY
```

As one might expect, the response time associated with these test were directly proportional to the data base size. The shortest table "scan" in these tests was through the PROFILE table when it contained only 3115 rows and the longest was through the ENTRY table when it contained about 980,000 rows. For the shortest, a response was obtained in about 1 minute 43 seconds, and for the longest a response of 3 hours and 42 minutes was required. Clearly, one must be leary of making queries of solely non-indexed fields with a large data base. The price in response time for not defining domains as indexed (or imaged in ORACLE terms) if they are to be used regularly as selection criterion is quite high.

Another observation which can be made from examining these two sets of results is that scanning the ENTRY table when it contained 90,000 rows took less wall clock time than accessing approximately 60,000 rows in the largest PROFILE table. Some difference might be expected because while processing the PROFILE table a comparison of the TAPE_ID field had to be made but this should not mean that 50% more rows could be scanned in less time. Probably the most likely cause is that when the PROFILE table has 60,000 rows the data base has about a million records while when the ENTRY table has 90,000 rows the data base is about one tenth that size. (Since all data is stored in the same VAX file or files the Profile and Entry table rows are interleaved within the file. When the entry file has 90,000 rows it occupied about 93% of the data base space utilized for tables so it is more densely packed into the data base. The Profile rows represent only about 7% of the utilized space requiring more disk reads to locate an equal number of rows since they are less densely apportioned in the file.)

Queries 5A, 5B, and 5C's function was to ascertain the significance of compound search criteria when soliciting information from the data base. The three queries were respectively:

```
SELECT COUNT (*) FROM PROFILE WHERE LAT = XXXX
SELECT COUNT (*) FROM PROFILE WHERE LAT = XXXX AND LONG = YYYY
SELECT COUNT (*) FROM PROFILE WHERE LAT = XXXX AND TAPE_ID =
"ZZZZZZ"
```

the search criteria in the first query is simply the indexed field LAT. The second compounds the search criteria by adding a second indexed field, LONG. The third compounds the first criteria by adding the non-indexed field, TAPE_ID, as a second qualifier. Thus, the first query serves as a control while the second measures the added cost of a second indexed field and, the third, that of a second non-indexed field.

The results of this set of queries are somewhat inconclusive for ORACLE. Perhaps the choice of LAT and LONG values should have been more selectively chosen so that a larger number of successful responses would have resulted. The values chosen yielded the following successful matches at the five ascending levels of data base size for the first or control query: 1, 2, 3, 9, and 28. The second query yielded only 1 successful match for each of the five levels and the third had the following responses: 1, 2, 2, 2, and 9 respectively. Comparing the successful matches with the response times of each of the three related queries lead to no consistent pattern of performance. (The times in the tables include the time to get all successful responses.) From these results it is assumed that no significant change in performance exists when changing the search criteria from a single indexed field to a compound one. Note should be made that no attempt was made to demonstrate the cost of choosing non-indexed fields exclusively in compound format. This would require a search of the entire data base similar to queries number 2 and 4 already discussed.

Queries 6A and 6B were intended to identify the cost associated with using the "ORDER BY" clause in ORACLE. This option, when invoked, returns the results of a query in a specified order based on the value of a particular field in the row selected. Query 6B was:

```
SELECT * FROM ENTRY WHERE PROFILE_CNT < 125 ORDER BY TEMP.
```

Query 6A served as the control run and was the same as 6B except it contained no "ORDER BY" clause. The number of responses to the query was 1,973. The control query completed in 81.08 seconds while the query requiring the sort completed in 93.11 seconds. The difference of 12.03 seconds is almost 15% more than the control query.

Queries 6A and 6B were only conducted once because the sort function is directly dependent on the number of responses not the total data base size. Several points should be made about the "ORDER BY" option. The use of "ORDER BY" when a "WHERE" clause references the same field and that field is indexed is a redundant procedure because the results of the query are returned in ascending order by virtue of the "WHERE" clause alone. It is unclear how the sort is internally managed by ORACLE and it may be possible that larger numbers of values to be sorted could overflow the available work space and create a large number of VAX page faults thereby degrading the sort response significantly.

Items 7A and 8A are intended to measure the incremental times to add a new row to the PROFILE and ENTRY tables respectively. The results associated with the insertion of a new row in the PROFILE table reveal that a direct relationship between size of data base and incremental load time required does not appear to exist. This defies the intuitive suspicion that the larger the data base the slower the performance but could be due to several factors. First, ORACLE load rates, as discussed in section 2.2.2, displayed remarkable consistency at all tested levels of data base size revealing a small variance in performance over the whole range of data base size. Secondly, because four of the five fields in the Profile table row are indexed the balance or state of the B-trees at the time of the insert may have considerable influence on the results. Lastly, the sample is for only five cases (only three of which are version 2.3 results) and a larger number of tests would have to be made to statistically assure that the relationship of data base size to incremental load rate is not significant especially in light of the small variance in load rates mentioned above. A similar observation can be made for query 8A although only one of five fields is indexed in the ENTRY table. The incremental load rates still show no pattern related to data base size. Comparison of the two does tend to demonstrate a consistent overhead related to the extra B-trees required to be updated in the PROFILE table as one might expect.

Comparison of the results of 7A and 8A with the load rates depicted in Figure 2-3 reveals that the incremental cost of an additonal row is much more expensive than the insertion rates indentified during the load process. This can be attributed to the cost of compiling the ORACLE insertion statement prior to loading. This is relatively signficant if only one set of values are to be bound for insertion but is neglegible when thousands of sets of values are bound for insertion.

Numbers 7B and 8B are intended to measure the impact of deleting a single row from the PROFILE and ENTRY tables respectively. The target rows in each table chosen for deletion were the ones added in number 7A and 8A. The deletion rates are all lower than the insertion rates but are somewhat proportional to the corresponding insertion rate. As a consequence, the results appear to demonstrate no relationship between data base size and incremental deletion rates. They also show that deletion of an ENTRY row with its single indexed field is consistently faster than deletion of a PROFILE row with four indexed fields.

2.3.3 SEED Test Results

The SEED benchmarking tests were conducted using version B11.3 at all levels of data base size. The results are provided in tabular form in Appendix I. Unlike ORACLE the primary terminal interface in SEED which is called HARVEST does not have all the capabilities available through the Host Language Interface (HLI). HARVEST only allows for data base queries and does not facilitate insertion, update, or deletion of records in the data base. A second Terminal Interface (TI) module called Garden is also available in SEED which permits exercising of all capabilities available through the HLI. For this reason some of the benchmark operations were performed using only one of the terminal interface capabilities. In general, however, HARVEST was exercised whenever possible because it is more user oriented and, as, such would be targeted for the casual user's primary interface. Where applicable,GARDEN was also exercised and in one case the report writer, BLOOM, was used.

Also, unlike ORACLE, there is a difference between the HLI implementation and the TI. ORACLE's HLI supports the commands in an identical string format to that for the terminal interface. SEED's HLI implementation is not equivalent to the HARVEST query language. It requires the establishment of currency by navigating the data base structure in a manner consistent with the function to be performed and then fetching or operating on the data as desired. The results reported in Appendix I for SEED's HLI and for GARDEN include the costs of navigation and currency establishment for each test. The logic implemented to navigate the data base was simple and straight-forward and significant improvement to the FORTRAN code was not considered possible so, for this reason, the SEED HLI results are not felt to be negatively influenced by poorly or inefficient written code.

The number 1 query in the test plan for SEED was:

```
WHERE P_TIME = "YYMMDDHHMM" DISPLAY P_TIME.
```

The actual time used was the last P_TIME value entered into the data base at the time of the test. The number 3 query was:

```
WHERE P_TIME = "YYMMDDHHMM" DISPLAY PRESSURE_LVL, TEMP
```

which verifies how effective the index capability is in SEED. Number 1 should require a "HASH" computation and the necessary I/O to fetch the particular record pointed to while number 3 includes the additional steps required to locate and fetch the entry level records that are associated with the desired P_TIME value (number 1 does not locate the LAT, LONG, and TAPE_ID associated with the P_TIME). Results for both of these queries indicate the indexing methodology is effective but some degradation is apparent at the 1,040,000 record data base level. This is probably attributed to index values "HASHING" to the same location and overflowing the physical page they would normally be stored on. As a result overhead develops because the overflow pages must be fetched and searched. Naturally with a limited amount of data base space (as in the "STATIC" approach used here, Section 2.2.3) more duplicate hits will occur as the data base grows unless an ideal hash computation is derived. It should be noted that this data base is only half as big at the one million record level as the planned for data base. Seeing evidence of degradation due to duplicate hits at this point is forewarning and points up the need for analyzing the data characteristics to determine a new user supplied hash code. Query 1 asks that a profile time be located and printed if it is available in the data base. Practically speaking one would normally expect to ask for additional information such as the LAT, LONG, and/or TAPE_ID. In ORACLE locating the Profile time implies that the LAT, LONG and TAPE_ID values are also located since they are on the same record. In the SEED schema used in this application more processing would be required to obtain that information. The Profile time value is located on a record that has pointers to three other records in another area (VAX file) which contain the LAT, LONG, and TAPE_ID values. These would have to be obtained in separate steps, if desired, once the Profile time was located. No measurements have been made to determining the additional costs of these steps, however.

Query number 2 requires that all Profile records be sequentially accessed and number 4 requires that all Entry records be sequentially assessed. The results of these queries demonstrate the cost of accessing non-indexed data or navigating through an entire set sequentially. Query number 2 and number 4 were respectively:

#2) WHERE DATA_TYPE = "FGGE LIMS" COUNT P_TIME and

Examination of the results indicate that response time is directly related to the data base size as one would expect.

It is important to note that the data base design is closely related to sequential access response rates. For example, in the data base design implemented for the benchmark testing, approximately 16 entry records are stored with an associated Profile record (see Section 2.1.2.2). A data base page contains 2 Profile records and about 32 entry records. The implication here is that a single physical disk read is required to retrieve 32 entries but only 2 Profiles (assuming a physical READ inputs a single data base page). The results bare this out by showing a response of over 295 seconds is required to access about 11,200 Profile records when the data base is at the 189,000 level while only about 250 seconds is required to access about 48,000 Entry records at the 50,000 level. If the design suggested for the "direct" loading in Section 2.2.3 were tested it would probably have had different results because Profile records would have been stored together on a data base page without the associated entry records. This would have significantly improved response time for accessing Profile records sequentially. In any event the time to sequentially access all records in a large data base is likely to be high and, if possible, the data base design should attempt to minimize the likelihood or need for performing such operations through the proper assignment of keyed fields.

Queries 5A, 5B, and 5C were intended to measure the effect of introducing compound selection criteria in the "WHERE" Clause. The three queries used were, respectively:

```
#5A) WHERE LAT = XXXX COUNT P_TIME
#5B) WHERE LAT = XXXX AND LONG = YYYY COUNT P_TIME
#5C) WHERE LAT = XXXX AND TAPE ID = "ZZZZZZ" COUNT P_TIME
```

As mentioned in the previous section, the number of responses to each of the above queries was relatively low and provide some inconclusive findings. Further, query 5C has a compound WHERE clause that uses both LAT and TAPE_ID. Although this is the same query as used with the ORACLE tests it does not accomplish the purpose of having an indexed and non-indexed combination because in the SEED application the TAPE_ID field is indexed while in the ORACLE application it was not. This oversight was not discovered in time and as a result no measurements were taken which reflect the use of a WHERE clause containing both an indexed and non-indexed field for SEED. The results of the queries do not indicate consistent results when comparing the use of two indexed fields versus the use of only one. They do indicate that compounding the WHERE clause with TAPE_ID results in faster response time than if compounded with LONG. The cause for this is probably due to the small number of occurrences of the TAPE_ID (from 1 to 8) versus the large number of longitude values (3,000 to 62,000).

Queries 6A and 6B were intended to measure the efficiency of the sort capability available in SEED's software. Number 6A was intended to be a control and 6B was intended to effect a sort for 1,973 temperature values in the entry record. Unfortunately neither HARVEST, GARDEN, nor the HLI currently support a sort option. The report writer, BLOOM, does have such a feature and this was used to produce the sorted results in 6B while 6A used BLOOM to generate an unsorted list. The use of BLOOM to produce sorted output is not suitable for an interactive user. It is not designed to be used like a query language and is much too procedural and complicated for casual users to use. See Section 3.7.3.6 for more information on BLOOM. These two tests were conducted at the 52,000 record data base level only. The response times associated with these two tests are a bit mysterious because the sorted output was produced faster than the unsorted. Examination of the CPU time required to produce the output does show an additonal 9 seconds were required to produce the sorted results. This is only about half of one percent of the total time required to produce the unsorted results implying that the sort adds relatively little overhead. As noted in the previous section it is not clear if larger numbers of values to be sorted may overflow the work space available to the sort thereby creating a significant degradation due to the inevitable page faulting that would occur.

Numbers 7A and 8A measure the incremental cost of inserting a new Profile record and Entry record respectively. The inserts consist of the DML commands necessary to obtain currency of the proper owner record occurrences and then to insert the new values. As was mentioned in the previous section, the times recorded are much slower than the load rates seen at corresponding levels of data base size but this is attributed to the initial overhead required to prepare to insert the record (i.e. to open the data base and estblish "currency"). With the exception of the results at the 52,000 record data base level the response times show an Entry record can be inserted at a much faster rate than a Profile record. This is most likely due to the way the Profile record is treated in the data base design, that is, the profile time, latitudes, and longitude values are all indexed or "CALC'ed" fields while the Entry record has no indexed values. (The anomaly seen at the 52,000 level is thought to be due to the choice of Profile time associated with the Entry record to be inserted. At that level a choice of time with approximately the same value as the largest time in the data base was made. Subsequent tests used a value with a significantly different value of Profile time than any in the data base. This may have resulted in page overflows when hashing to the Profile time value at the 52,000 level that was not present at other levels). Examination of the results also shows that data base size is weakly related, if at all, to the speed of the insert. This would tend to contradict the results of the loading process discussed in section 2.3.3,but a closer examination of the inserted values in number 7A and 8A may explain this anomaly. The values chosen for LAT, LONG, and P_TIME were all extreme so that their subsequent deletion would not cause other Profile records to also be deleted. The values chosen probably "Hashed" to clean pages in the data base making the insertion somewhat independent of the data base size. This again underlines the importance of the hashing algorithm and its relation to data base performance. The Entry record inserted in 8A while not containing indexed fields is associated with the profile record that was inserted in 7A and therefore benefits from the hashing to the profile record when establishing currency before inserting the entry data.

7B and 8B measured the cost of deleting a single Profile record and Entry record, respectively. SEED's approach to deletion is to mark the relevant occurrence as deleted but not to remove it until all the pointers associated with it have been properly updated. This means a minimum of work is done during the delete procedure and examination of the results reflect that. It is also apparent that data base size has very little relationship to the response time required for the delete. Note again should be made, however, that the values deleted were probably "hashed" onto or related to values "hashed" onto "clean" data base pages as discussed for the insertions in 7A and 8A above. Hashing cleanly to the data without overflowing would make the delete response independent of data base volume. From a comparison of the results of inserts 7A and 8A and the load rates at corresponding data base levels it seems likely that the deletion of a range of values that are chained contiguously would be faster per deletion than the single delete rates obtained in 7B and 8B.

2.3.4 RIM Test Results

The queries and commands chosen for the RIM benchmarking tests were patterned after the ORACLE queries for the most part, but because P-TIME was not an indexed domain after the 52,000 row level some of the queries had to be modified to use the indexed domain PRO-CNT instead. There also existed several problems which prevented some tests from being executed, but these cases are confined to the incremental deletion and insertion tests. Queries 7A and 8A, insertion of Profile and Entry rows, respectively, could not be done with the TI due to a RIM software error. Queries 7B and 8B, deletion of Profile and Entry rows, could not be done with the HLI due to a linkage problem of unknown origin with the RIM software.

Because ORACLE and RIM are based on the relational data base model and since their applications share a common set of table and domain specifications a tendency to make direct comparisons of the two is natural, but some considerations should be made before doing so. One must recall that the ORACLE results have to include the detached process overhead while RIM does not. The TI results of both systems include overhead required to Open and Close the data base as well as the costs of the function being tested. Also it is important to remember that RIM could not continue in the testing with Profile time as an indexed value so that the first test had to be modified and the inserts and deletes have one less index field in the Profile Table to be concerned with.

The number 1 query for RIM was originally:

```
SELECT P_TIME FROM PROFILE WHERE P_TIME EQ "YYMMDDHHMM".
```

The measurements derived from this query were to assess the effectiveness of the RIM indexing capability using the smaller PROFILE table. In section 2.3.4, RIM Load Results, an explanation of the problems of indexing this field was given. As a result of the problem the number 1 query was altered for all succeeding data base levels above the 52,000 row size to:

```
SELECT P_TIME FROM PROFILE WHERE PRO_CNT EQ XXXXX.
```

Examination of the response time for this query while using P_TIME as the search criteria at the 52,000 level reveals the severe penalty associated with the indexing of this twelve character string. Once the PRO_CNT field is substituted the responses show an effective indexing capability which appears to suffer little degradation as a function of data base size.

The number 3 query was similar in purpose to number 1 but was designed to exercise the indexing capability using the large Entry table. The query used was:

```
SELECT ALL FROM ENTRY WHERE PRO_CNT EQ XXXXX.
```

An examination on these results shows agreement with those in number 1 and leads to the belief that the indexing technique used by RIM is effective for numeric data. Because of the approach taken, which only uses the first four bytes of the field for indexing, an inefficiency will result when indexing many values greater than four bytes in length whose first four bytes are identical.

Query numbers 2 and 4 were both designed to access all of the rows in the PROFILE and ENTRY tables, respectively. The queries chosen for this function were:

```
#2)  SELECT P_TIME FROM PROFILE WHERE TAPE_ID EQ "XXXXXX"
#4)  COMPUTE COUNT PRO_CNT FROM ENTRY*
```

(The "XXXXXX" in #2 is used to force a search through the entire table.) In both cases the results are as expected and show a very linear relationship between data base size and response time. The shortest response time was just under 30 seconds and is related to the accessing of 3115 rows in the PROFILE table. The longest response was 16.5 minutes counting the PRO_CNT values in the ENTRY table when it contained about 980,000 rows. RIM's speed in searching the entire data base is considerably faster than that of either ORACLE or SEED.

* Originally query 4 was SELECT ALL FROM ENTRY WHERE PRESS_LVL EQ "X" AND PRESS_TY EQ "Y" AND TEMP EQ "ZZZ" AND QC_FLAG = M but was changed after the 99,000 row level to just count the rows.

Queries 5A, 5B and 5C were designed to measure the cost of compound search criteria. The three queries used were respectively:

```
#5A) COMPUTE COUNT PRO_CNT WHERE LAT EQ XXXX
#5B) COMPUTE COUNT PRO_CNT WHERE LONG EQ YYYY and LAT EQ XXXX
#5C) COMPUTE COUNT PRO_CNT WHERE TAPE_ID EQ "ZZZZZZ" and LAT EQ XXXX
```

The first query simply used a value of LAT, which is an indexed field, as the search criteria and thereby serves as the control measure. The second adds another indexed field and the third adds a non-indexed field to the search criteria. As stated in the previous two sections, the values of LAT, LONG, and TAPE_ID chosen yielded a relatively small number of responses which made it impossible to draw valid conclusions from the measurements recorded. The results at varying data base sizes are conflicting when all three queries are compared. When the Profile Table is largest (60,000 rows) the HLI results show that an indexed and non-indexed combination requires 20% more time than the control while using two indexed fields requires more than 75% more time. When the Profile table contains 26,000 rows, indexed and non-indexed, and both indexed require about 25% more time than the control run. With the Profile table containing its lowest level, 3115 rows, the control is slower than either of the others by .02 and .04 seconds, respectively. From the results available, no consistent patterns can be detected nor are any serious degradations apparent.

Query 6A and 6B were designed to measure the cost of requesting results in a sorted order. Query 6A was a control while 6B retrieved the same data but in a specified order. The two queries were:

```
SELECT ALL FROM ENTRY WHERE PRO_CNT LE XXX
SELECT ALL FROM ENTRY SORTED BY TEMP WHERE PRO_CNT LE XXX
```

The queries were only exercised at the 52,000 row data base level and a PRO_CNT value was chosen that would require the sorting of 1,973 values. The results of the queries show the sorted requests required about 16%

more time for the HLI test and about 9% more for the TI test. It should be noted again that documentation of the internal functioning of the RIM software was not available and there may be a point where an internal work space used for sorting is overflowed causing a significant degradation in sort performance due to page faulting.

Queries 7A and 8A were intended to measure the incremental time to insert an additonal row to the Profile and Entry tables respectively. The results associated with inserting a row into the Profile table show a direct relationship exists between data base size and response time required for insertion with the exception of the first result. The exception is due to the previously mentioned problem associated with indexing the P_TIME field. At all subsequent data base levels only the LAT, LONG, and PRO_CNT fields have been indexed. Query 8A showed no pattern to indicate that the insertion rate for it was related to data base size although at the 1,040,000 row level over 2 seconds was required while previous levels had required less than .25 seconds. The difference noted between 7A and 8A may be due to the fact that insertion into the ENTRY table requires a single index be updated while such an operation for the PROFILE table requires three updates. In any case incremental insertion into the ENTRY table is signficantly faster than that for the PROFILE table. A serious failure was discovered in RIM while performing these insertions. The TI could not perform the insertion function and had to be omitted from the test procedure. It was subsequently discovered during the performance of 7B and 8B that the HLI could not be used to delete rows from the data base and the HLI tests were omitted from 7B and 8B.

The delete operations, 7B and 8B, could only be conducted with the TI. To estimate the net time required to perform these tests (i.e., to remove the open and close data base overhead) one can subtract out the results of test number 9. This can also be done for the other tests conducted using the TI. However, the variance the number 9 results may have with that of a particular test is unknown and erodes the validity of the net result. In one case a negative time was generated using this approach underscoring the variability which exists.

3.0 QUALITATIVE ANALYSIS

3.1 User Friendliness

3.1.1 ORACLE

The relational model which ORACLE is based on is considered by many to be the most comprehensible data representation currently available. The concept of two dimensional tables is easily understood by most potential users especially the target users in the NASA environment. Normally, the relationships among the data are readily apparent to a user when the tables are presented. In most cases it is quite easy for a user to design tables that will accommodate the application without studying a great deal of data base theory. If one can establish a "first normal" form for the data, one can produce an acceptable data base design for ORACLE to manage. This does not mean an optimal design is inherent, but it does imply that a relatively casual understanding of data base concepts is required, which permits the user to focus his thoughts on his data instead of on data base intricacies. Certainly complex data structures and medium-to-large-scale DBMSs would benefit from, if not require, a data base specialist to optimize the design and avoid serious pitfalls.

The primary interface to ORACLE is the SQL language. SQL embodies all the Data Descriptive Language (DDL) capabilities as well as the Data Manipulation Language (DML) capabilities. It is used as the input language for the terminal interface called the User Friendly Interface (UFI) as well as for statements represented as character strings in high level languages which communicate with ORACLE through the Host Language Interface (HLI). The use of a single language simplifies the learning process and normally makes it easier to locate descriptions about a particular syntax or capability because a single document can be referenced.

The language itself requires a defined syntax. It is currently lacking a "help" function or other user friendly guide to aid a user in constructing DDL or DML statements. When using UFI, the user is given an indication of the location of the first field which causes the SQL statement to fail when an error is present. The user is not notified of other errors which may be present at the time. The error identified is described briefly, but accurately, in a "canned" phrase. The user may take advantage of the UFI editor to modify the most recent SQL statement and resubmit that statement without retyping it, which can save time and minimize additional typographical errors. Using the HLI, errors are identified by status codes and are pointed to by an offset from the first character in the SQL statement. In general the infrequent user may find it unacceptable to use UFI because of the syntax requirements and lack of tutorial assistance. The incorporation of a "help" function is needed to provide the infrequent user with a bridge to understand the syntax and refresh his memory.

Associated with a data base in ORACLE are system tables and views of system tables which comprise the systems knowledge of the data base. This information decribes a data base's tables, views, domains, and domain attributes which constitute the information pertaining to a data dictionary. Also provided in these tables are definitions of user privileges defined for the tables in the data base and definitions and descriptions of the dictionary tables themselves. A user may survey this information to gain knowledge about the data base design and structure, about a particular column or domain's use or occurrence in the data base, or about his privileges to access or modify information in tables in the data base.

ORACLE has implemented an approach to naming domains or columns without regard for consistency between tables. For example, if the domain, "TIME", in table A was defined as a character field that was stored as hour, min, and second ("HHMMSS"), a second domain called "TIME" may be present in table B which is a numeric field stored in total seconds. The fact that the domain, "TIME", exists in both tables does not imply that both represent the same logical entity. If consistency is desired, the user is given responsibility for establishing and enforcing rules to maintain it. Without user enforcement, an ambiguous and confusing environment can evolve.

ORACLE can provide a user with summary results about the data in the data base. This includes a count of responses which meet query requirements as well as providing minimum, maximum, average, and sum values for fields of rows which meet the query requirements.

3.1.2 SEED

THE CODASYL Network model which SEED is based upon requires a schema to define the data base structure. Within the schema description lies the relationships between the data in the data base. The concepts of one to one, one to many, and many to many relationships need to be understood and applied appropriately in the schema specification. The user must comprehend the hierarchy of his data so that a suitable data base structure can follow. The formalization of these relationships for the production of a data base design is normally unnatural to a user unless he has had prior experience with a CODASYL system. The user will, in many cases, be required to understand the system of pointers which chain together occurrences of his data so that he can navigate the data base structure to perform desired operations. The exception to this is the terminal interface language called HARVEST which navigates the data base structure for the user. The definition and compilation of schemas and sub-schemas and the navigational considerations imply that the user must divert a significant amount of attention away from his primary concern, the data, to understand how to use SEED for his application. It certainly means that a data base specialist would be required to analyze requirements and design and define structures for all but the simplest of applications in the NASA target environment.

There is no primary interface language with SEED. As mentioned above, the HARVEST terminal interface permits the user to access data in the data base without need to navigate the data base structure. It is limited to query capability and, as such, does not possess the power to insert, delete, or update data. The language itself requires a defined syntax, but does possess a "help" function to guide the user through command construction. The other terminal interface in SEED is GARDEN.

GARDEN provides the full compliment of Data Manipulation Language (DML) commands. It also provides a method to get information about a particular data base's sets, records and items and contains a "help" facility to instruct a user in how to use the DML. To use GARDEN one must understand the data base structure and be knowledgeable about navigation and currency. When interfacing with SEED using a high level programming language like FORTRAN a usér must use a DML that is similar to the commands used with GARDEN.

The formality inherent in the CODASYL model and the need to navigate the data base structure are barriers to most casual data base users. The HARVEST interface relaxes these demands, but facilitates queries only and "points out" errors by cursor position without describing the nature of the error. The user is forced to continue the input sequence from the point of the detected error and must escape from it with a control character sequence if he opts not to continue the current input line.

Both GARDEN and HARVEST provide the user the ability to request information about the areas, items, records and sets defined in the sub-schema that is currently active. This provides a data dictionary capability to aid the user in understanding the content of the data base as well as the relationships between the information. The actual Data Description Language (DDL) defining the schema or sub-schema may provide a better understanding of the structure of the data base design and can be obtained by examining the source files used to define them or via the SCDUMP utility.

SEED permits a user to request a count of successful responses to a particular query without getting the actual responses. It also permits a user to request summary information including minimum, maximum, average, sum, and standard deviation for fields successfully meeting query requirements. GARDEN provides a capability to produce simple graphs and histograms from data in the data base as well.

3.1.3 RIM

The RIM system is a single-user data base management system (DBMS) based on the relational model. In a relational model users are only concerned with designing the logical structure of a data base; they are not burdened with a data base's physical design. Most users will find it easy to design and query RIM data bases. The schema of a data base must be declared in the DEFINE submodule. A data base schema defines a set of flat tables or relations, a set of table columns or fields, the constraint rules on fields in tables, the owner password to the schema, and read or modify passwords to the tables.

An interactive user need only learn the DEFINE submodule, the LOAD submodule, and the RIM query commands to know the Data Descriptive Language and Data Munipulation Language capabilities. A HELP function in RIM provides a description of these RIM commands and submodules, a summary of the syntax of each command, and a description of RIM "where" clauses. Therefore, the learning process is easy and the user always has information available if he forgets the syntax of a command. Users may enter RIM commands in a free-field format, therefore multiple commands may be entered on one line separated by a semicolon. RIM remembers each previous command so that all or part of the previous command can be re-used.

RIM commands provide access to all tables, selected rows and/or selected columns in a table, and a combination of information from different tables. Functions exist to find the minimum, maximum, average, sum or count of a column. The function that finds the count of a column operates on all types of columns (i.e. real, integer, or text). The functions which find the average and sum of a column operate on real and integer type columns. The functions which find the minimum and maximum of a column operate on all types of columns that are eight bytes in length or less.

A user may access the Data Dictionary with three commands: LISTREL, EXHIBIT, and PRINT RULES. The Data Dictionary provides users with information about the tables in a data base, characteristics of the fields in a specified table, the date that a particular table was last modified, the existence of read or modify passwords on a table (the actual password is not shown), the current number of rows in a table, and all the existing constraint rules on a table.

The RIM DBMS maintains consistency with the schema in the DEFINE submodule. Columns with the same name must have the same data type. In the LOAD submodule, if a constraint rule is violated, the DBMS displays the constraint rule violated by the user.

3.2 Flexibility

3.2.1 ORACLE

The ability to add more data to the ORACLE data base in the form of additional tuples or rows is not a problem. If the space allocated at any point becomes insufficient, an extension can be added to provide additional data base storage. The number of extensions is said to be unlimited, but in section 2.2.2 a case is described where multiple extensions may have caused a data base failure. This was said to be related to version 2.2 and is supposed to be corrected in version 2.3. The final data base used in the quantitative study for ORACLE had three extensions and performed without problem. Extensions offer a dynamic and flexible approach for defining the data base size to meet current needs without having to allow for future growth until the growth occurs. The growth can be related to more tuples in an existing table or to a change in data base structure.

ORACLE permits a user the flexibility of redefining a data base to accommodate differing needs without paying a significant price with his existing data. New tables can be added to the data base without impact to existing tables. New columns (domains) can be added to an existing table with any possible combination of attributes (i.e., imaged, unique, null, type). Once a new column has been defined, the user is responsible for updating the tuples accordingly so that the appropriate values are associated with the new column. There is no capability to delete a column from a data base table, although this is promised to be available in version 3.0. If the column permitted null values, it could be eliminated by assigning the null value to the field in each tuple, and then a new user view could be provided that did not include that field. The result would e a table that used no space for the field (since nulls are represented by default), and the user would not be aware of the field's existence. There is no capability to redefine the attributes of a field. RSI has indicated that a future version of ORACLE will permit indexes to be created or dropped dynamically for fields to give greater flexibility (probably version 3.0). All fields can be updated if required.

The use of views mentioned above also provides the flexibility of adding columns to the data base without affecting existing users or programs that do not need the new information. This logical data independence can be of great importance in reducing maintenance costs for old programs in an evolving data base environment. Views do add overhead for the system to retrieve the view definition, to effect the intersection of tables, if necessary, and to build the desired response. The amount of overhead is related to the view definition's content and complexity. In general, the ORACLE system is permissive to a change in structure or size without requiring a great deal of reworking of steps already done. This is not to say that in all cases changes are painless because some alterations require unloading and reloading the data base.

3.2.2 SEED

SEED requires that a user define the data base size prior to the insertion of any data in it. He may, however, define a data base as "Dynamic" which permits the data base to grow larger as it is filled. If he doesn't choose to use the "Dynamic" approach, the eventual data base size must be estimated accurately because too big a space will waste storage media and too small a space will require the unloading and reloading of the data base to recover. If the "Dynamic" approach is used new data base pages are allocated when overflow conditions occur. One must be careful because overflows can occur when there are available pages elsewhere and the data base grows dynamically anyway. It is not clear whether all the dynamically obtained pages must be searched or traversed when an overflow occurs to find where a new entry may go or if a new page is fetched outright. The former would require additional processing time and the latter might waste storage space. In any event, the overflow pages would require some additional disk I/O which would not ordinarily occur.

The schema definition in SEED dictates a rigid structure for a data base design. Normally a modification to the data base design results in a costly procedure requiring the unloading of all or part of the data base, the respecification and recompliation of schemas and subschemas and the reloading of the data base. If future changes can be anticipated a design is sometimes plausible which limits the chances to a single area thus requiring that only a part of the data base (that area) be unloaded and reloaded although the schemas and subschemas would still have to be respecified and recompiled. One approach to the problem when an anticipated design odification will require the definition of new records is he use of dummy records in the original schema. In his approach a dummy record is inserted for each owner record and the schema is defined so that the dummy records reside in a separate area in the data base. When the modification is defined a procedure that requires the unloading of the dummy area, the respecification and recompilation of the schema (only updating the record description just determined) and subschemas and the reloading of the dummy area with the new data corresponding to the new record description is needed. The implication here is that if one can initially foresee where the need for changes will arise then he may be able to design a structure which minimizes the impact when they occur.

The concept of sub-schema access can insulate some users and programs from certain types of changes in the data base design thus providing some logical data independence. The sub-schema can provide a program or user access to only the records and sets he needs to know about. Therefore, if the data base is modified and the information relevant to a user is not structurally modified that user is insulated from the change through his subschema.

In general it must be stated that the CODASYL model and the resulting SEED implementation is not designed to accommodate frequent changes to a data base design without imposing significant overhead. A thorough analysis should proceed the design of any data base to understand fully the characteristics of the information to be managed and the needs of the user community. The chosen design will then require little or no alteration and where modifications are required that were anticipated the procedure mentioned earlier may minimize the cost of the changes. In instances where initial data characteristics or user needs cannot be fully identified and defined, one might expect that future data base changes will be required and will prove to be costly with SEED.

3.2.3 RIM

The RIM DBMS dynamically allocates more space for a data base as it increases in size. If there is not enough disk space available for this dynamic allocation, the data base cannot be enlarged (RIM does not provide the capability of storing one data base on more than one disk). Therefore, the user must always be aware of the amount of available free disk space. It is the user's responsibility to recover unused data base space by issuing the RELOAD command. This is necessary because the space of a deleted table or row cannot be re-used until a RELOAD command is executed. When the RELOAD command was executed (in version 4.0) after the deletion of a row or a table, the DBMS displayed an error message:

```
-ERROR- on unit 52 with status 25.
```

From the testing that was performed it could not be verified that the RELOAD command functioned properly nor was the source of the error message determined.

A RIM data base schema can be modified in a number of ways without redefining the entire data base and reloading the data. New tables can be added to a data base. Existing tables can be deleted. Columns can be added to or deleted from a table. Table or column names can be changed. Relational algebra commands (e.g., INTERSECT, JOIN, PROJECT, and SUBTRACT) can be used to create new tables from old tables. The BUILD KEY command can be used to change a previously defined unindexed column to an indexed column. The DELETE KEY command will change an indexed column to an unindexed column. Passwords to read and/or modify tables can be changed. The constraint rules (section 3.4.3) declared for a table and its columns cannot be changed. However, during the interactive load of a table the user can specify that these checks not be made.

3.3 Host Language Interface

3.3.1 ORACLE

ORACLE supports an interface between it and a number of programming languages including: FORTRAN, COBOL, PL-1, "C" and the VAX native mode instruction set (using macro instructions). The interface, termed the Host Language Interface (HLI), permits the full use of the SQL language including query, data manipulation, data definition and data control facilities. The actual SQL statements used in the interactive interface are input as character strings for the HLI to compile. The HLI uses work areas defined in the users program area for communication between ORACLE and the user. Steps the user's software can make with ORACLE include:

- "log on" (required)
- open a "cursor area" (required)
- request descriptions of data base fields
- define fields requiring conversion for internal representation
- "bind" values to fields in a SQL statement
- execute the current SQL statement
- repetitively "fetch" rows of output
- "log off" the data base (required)

Some of the above steps are not necessary in certain instances and more than one "cursor" may be active so multiple SQL statements can be executable at a given point in a program.

The fact that the entire set of SQL capabilities exists through the HLI implies that software surrounding the ORACLE system can be produced to accommodate many unique needs a particular application may require. Most commercial DBMS packages cannot be expected to address these needs. The proposed Packet Management System (PMS) being developed by NASA for the NEEDS Phase II program required among other things the management of packetized header data received over high speed data lines, management of various data sets now stored as sequential files on magnetic tapes, and management of catalog information about data maintained in an archive. None of these requirements are directly supportable by ORACLE, but with the HLI, software could be built around ORACLE to accommodate these needs. The HLI does not constrain the amount of software which can surround the DBMS, so the primary limitations are system bounds and fiscal budgets. Many of the functions stated as requirements for the PMS are not explicitly met by any commercially DBMS. The functions are either too unique to the application or too specific for a general purpose DBMS to meet directly. The HLI capability in ORACLE permits software to be added around the data base software to accomodate the desired customization to meet system goals.

The creation of an ORACLE data base requires the running of the DBF utility which requires interactive input for the specification of the data base. This step would make the generation of a data base by means of software a difficult problem and is not a recommended approach. The implementation of generic software which could access tables originally unknown to it is conceivable but would require explicit rules relating data in one table to that in another for the production of such software to be of use. It would also require the overhead of searching the system tables in order to determine the contents of the data base and this approach may be too expensive in terms of response time.

3.3.2 SEED

SEED supports an interface between it and both the FORTRAN and COBOL languages. The interface, referred to as the HLI, permits software to fully navigate the data base to access, insert, update, and modify data within the limits of the subschema specified by the calling program. The HLI is employed using the same approach as used with GARDEN. In this case the software must navigate and establish currency instead of a user at a terminal. For FORTRAN, a sub-schema must exist and must be processed by the SEED utility SUBFDP which generates a user work area (UWA) that must be inserted at the beginning of the user's source program. This area will facilitate the communication between the user's code and SEED. The user's code must "open" some or all areas of the data base, navigate the data base accordingly to establish currencies, retrieve, update, insert or delete information in the data base, and "close" the data base.

The HLI provides an interface for user software to fully exercise SEED's capacity to manage data. For those applications such as the proposed PMS, which have highly unique and specialized needs, no commercially available system can be expected to satisfy all system requirements. With the HLI a user can provide the customized software that complements SEED's DBMS capabilities and meets the requirements of the application. There are no bounding constraints to the user's software when interfacing via the HLI, so the primary constraints to surrounding SEED with software are system bounds and fiscal budgets.

The generated User Work Area (UWA), for FORTRAN, identifies and names the variables which the software will use to communicate with SEED reducing the programmer workload. Also the GARDEN module provides an inherent mechanism whereby a programmer can become familiar with the same data manipulation language (DML) used in the HLI and the navigational steps required for a given data base design and operation. This can facilitate the logical debugging of the interface software in an interactive mode to reduce the programmer effort. The procedural nature of the schema and subschema definitions and the use of the respective compilers, FDP and SUBFDP, in preparation for using the HLI makes the dynamic definition of a data base through user software very difficult. The production of generic software that could navigate and access different data base structures originally unknown to it is not a practical or realistic consideration for a system such as the proposed PMS.

3.3.3 RIM

RIM data bases may be accessed and modified by application programs through FORTRAN-callable interface routines. The programming language interface supports the following two operations:

1. moving a row from a data base to an array (supplied by the application program)

2. moving a row from an array (supplied by the application program) to a data base.

These two operations can be used to access, modify, and load data.

To move a set of rows from a data base to an array supplied by the application program, the application program must let RIM know which rows are desired and get the desired rows. The RIMFIND and RIMHUNT interface routines enable the user to specify a selection criteria for retrieving a set of rows from a table in the data base. The set of rows retrieved may be sorted using the RIMSORT interface routine. To retrieve data from the table for the desired set of rows, the RIMGET interface routine is used. RIMGET puts the data retrieved into the array supplied by the application program.

To modify a row after retrieving it from the data base, the RIMPUT routine is called. The application program supplies the array in which the modified row is placed and passed to the data base through RIMPUT. To load new rows in a RIM data base, the RIMLOAD routine is called in which new rows to be inserted are passed.

Not all RIM capabilities are available through the Host Language Interface that are available through the interactive on-line mode (e.g., tables cannot be created or expanded; columns cannot be deleted, and relational algebra commands cannot be executed). Therefore, this programming language interface would be more powerful if all RIM capabilities were available. As shown by the quantitative analysis of RIM, the programming language interface can effectively be used to load and query the data base.

3.4 Control

3.4.1 ORACLE

The concept of a centralized Data Base Administrator (DBA) responsible for the overall use of the data base was considered a requirement for the NASA target environment. ORACLE does not currently support such a concept. Any user may use DBF to create a data base and then has the power to explicitly grant other users access to the data base (with the SQL command DEFINE USER). Once given access to the data base a user may in turn give other users access. A data base creator also has the option of allowing all persons access. Once a user can access a data base, the user may create his own table(s) and may identify which other users have the right to access the table and how they may use the table including: read, insert, delete, update and expand. A user may also be given the privilege to grant his own privileges to other users.

The implication is that control exists but not through a central individual or group. This would have to be accomplished through a set of operational standards. The LIST option available in the DBF utility does identify the data bases which are currently defined to the ORACLE system data base as well as the number of blocks in each of the data bases' constituent extents. A DBA would not be able to learn more about a data base if it was secured and he was not granted access to it. (If a data base is secured, the LIST option will identify the original creator's name, however.) Since one must log on the VAX system under the ORACLE account before starting the ORACLE system, centralized control can be maintained over the activation of the DBMS since the account is password protected. This is also true of the procedure required to bring the ORACLE DBMS "down" gracefully as well.

Also of note is the lack of control of disk space. Any user may, at any time, attempt to enlarge the data base using the DBF utility. The only constraint is the available contiguous space on the storage medium. The potential thus ex sts for a user to consume all available space at the expense of other data base applications which have higher priority but which cannot find space needed for their data.

Related to the idea of control is the presentation of only pertinent information to users. This includes the omission of information that is either superfluous to a given type of user or too sensitive to be made available to all users. ORACLE can address this need through the use of "views" that include only the "need-to-know" information for a class of users.

3.4.2 SEED

The concept of a DBA is not supported in the methodology adopted in the SEED DBMS. The DDL describing the schema permits the originator the ability to define a password to limit other users from accessing it unless they are told the password. Likewise the sub-schema DDL can use the schema's password or a new password to control access to it. This password protection does not facilitate the DBA concept, however. Implementing a DBA controlled system would require the use of operational procedures that compels potential users to request use of SEED prior to using it. The use of DBSTAT results in the identification of the logical "areas" of a SEED data base but not the actual VAX file specification and location. No direct capability exists to identify what VAX files are SEED data base files. This limits the central control of the DBMS.

The control of data base size and therefore the consumption of the available mass storage is defined by a user in the schema DDL. Since any user could define a schema and run DBINIT there is no explicit central control of the use of disk memory for SEED data base applications.

The use of sub-schemas to control or restrict the access of information in the data base for a class of users is effective. Sub-schemas can protect parts of the data from access by users who do not need or should not be permitted to view them. Logical areas may be omitted from a user's "window" to the data base all the way down to a specific item or field within a record.

3.4.3 RIM

The concept of a DBA is not supported by the RIM DBMS. The creator of a data base assigns the schema a password. Only the users knowing this password can change the schema through the DEFINE submodule. The DE INE submodule allows a read and/or write password to be placed on each table. This read/write password can be changed by anyone knowing the schema password. No command or utility is availabl to list all the passwords corresponding to a data base. Centralized control of a data base is not available. Centralized control of all RIM data bases is also not available; there is no capability to find all RIM data bases and their locations.

The DEFINE submodule provides three types of constraint rules on tables. A column in all tables may be constrained to a set of values; a column in a specified table may be constrained to a set of values; and two columns in the same table may be compared to each other with a certain rule (e.g., = , = , > , $\geq$, < , $\leq$). At anytime during the LOAD submodule, which loads rows on-line to one or more tables, a user may specify with the NOCHECK command that the rules stipulated in the DEFINE submodule are not to be checked. The CHECK command specifies that each row is checked as it is loaded.

3.5 Security

Security is closely related to the preceding section on control and some capabilities exist in the systems that overlap both subject areas.

3.5.1 ORACLE

As mentioned in Section 3.4.1, ORACLE offers the creator of a data base the right to restrict use of his data base to himself, to the "Public", or to specific users defined individually to ORACLE. (Currently if a data base is defined for "Public" use and as "READ" only, users are still able to insert, update, and delete anyway!) Initially when defining the data base the user identifies himself and his password to secure the data base. Then using the DEFINE USER command he identifies the users that can have access to the data base. Any legitimate data base user may create and define tables and identify users who may access them as well as the privileges each user may have including: READ, INSERT, UPDATE, DELETE, EXPAND, and if that user may grant his privileges to still other users. Users must then know their passwords to sign on the data base and also must have been granted rights to access the table(s) they wish to operate upon. A user's privileges may be revoked or changed by the grantor of those privileges or by the individual who granted privileges to the grantor and so on. Data base users identified by a DEFINE USER command must identify themselves and their passwords in order to "Log On" to the data base.

It must be mentioned that although ORACLE permits the securing of information under its management the data is stored in VAX files. In a secure environment, steps would have to be taken that would assure the integrity of the VAX file. The data base creator is considered by the

VAX operating system to be the owner of the associated file(s) and as such he must control access to his files. The most protection available appears to be the limiting of access to all users with the same UIC group specification. For the PMS environment the ORACLE security is probably adequate because the data is truly not sensitive. In a secure environment, however, the VAX files would have to be managed to prevent access to even hexadecimal dumps which could be decoded.

Also related to security is the ability to give different users different accesses to the same data base. As stated above each table in the data base can be treated independently in terms of defining user access. If a user or users need access to a portion of a table but should not be permitted access to all of it a "View" may be defined which will present only that information specified. This enables sensitive information to be excluded. The "View" provides for vertical exclusion as well as horizontal exclusion. This means that specific columns may be excluded and also that rows can be included or excluded based on defined conditions.

3.5.2 SEED

SEED's primary method of securing the data base is the password specification allowed for both schema and sub-schema access. Since all direct access to the data base through SEED requires a schema and sub-schema the data base is secure. Access to the VAX file that contains the data base or to the files containing the DDL and the passwords associated with the schema and sub-schema(s) must be protected by the creator of those files using the VAX file management capabilities. The compiled DDL has passwords encripted but the source does not. A secure application may wish to require the deletion of the source once it is compiled to prevent undesirable access to the passwords. As stated in the previous section, a truly secure application would have to prevent dumps of the data base files that might be decoded.

The use of sub-schemas can prevent users from accessing information in the data base which is sensitive and not meant for their use. The sub-schema can prevent knowledge of or access implicity to whole SEED areas or explicitly to records, subsets of records (fields), set and item types. Access denial based upon conditional values is not supported through sub-schema specification.

3.5.3 RIM

RIM offers password security to a data base. The schema cannot be re-defined unless the owner password to the schema is known. All tables may optionally have read passwords and modify passwords. Read passwords can be made known to a select group of users if certain tables contain sensitive information. Only users responsible for updating tables should have access to modify passwords. If two tables are given different read passwords, they cannot be used together. For example, if table A has 'BLUE' as the read password and table B has 'WHITE', the user must set the current password to 'BLUE' (i.e. USER BLUE) to read table A but table B cannot be read. To read table B, the user must set the current password to 'WHITE', but table A cannot be read. If table B is not given a password, then once the current password is set to BLUE both table A and table B can be read. Therefore, care must be taken in defining passwords. There are no capabilities to assign passwords to columns in tables.

A RIM data base is made up of three VAX files. The creator of the data base can protect these files using the VAX file management capabilities. It is good practice to have a backup of these files in case of integrity if the data is lost.

3.6 Processing Consistency and Recovery

This section addresses the DBMS's capabilities to insure data consistency, to resolve contention in the case of multiple accesses to a given piece of information, to recover from failures and/or errors, and to back up the data base.

3.6.1 ORACLE

Numeric data introduced to ORACLE has several limitations beyond that imposed by the VAX/VMS system. Although the VAX permits the storage of an integer as large as 2,147,483,648, ORACLE accepts integer values greater than 9 digits in length but displays the information in exponential notation (Version 2.3 may change this). This causes a problem when a subsequent query is made with a "WHERE" clause that uses the exponential form because ORACLE does not identify its displayed value as being equal to the internal value. ORACLE will identify the original integer value as equal to the internal value but that integer value may not be available after its insertion into the data base. The use of a "UFI FORMAT" declaration can relieve this problem for insert but is too procedural and unfriendly and isinvokable only from the interactive interface. It also appears that real numbers presented to ORACLE via UFI can be referenced exactly as they are input while the same real numbers can lose some of their accuracy when introduced via the HLI.

ORACLE "locks" a row when one person updates it to prevent dual updates from overwriting each other. This function is supported automatically without user request. To obtain a higher level of "lock-out" the user may employ the BEGIN TRANSACTION and END TRANSACTION commands which explicitly prevent the concurrent update of a specified table by other users while the user is performing his transactions. The BEGIN TRANSACTION has the option of locking other users out completely or just from update commands.

The Automatic Row Lockout Feature will prevent the occurrence of the "Deadly Embrace" phenomenon. That is concurrent users would not be able to begin multiple row updates during which each user finds that the other has a row "locked" that the other needs in order to complete the update. If two users attempt to issue a BEGIN TRANSACTION on the same table, the request received second is suspended until the first requested issues an END TRANSACTION for the table. The suspension could cause a problem especially in an HLI application. Without an indication of why his BEGIN TRANSACTION request has been suspended, the interactive user might escape from his suspension by issuing a "Control/C" sequence. He could then repeat his request or go on to another function. An HLI application does not have this latitude and would be suspended until the END TRANSACTION was issued by the other user. An implication here is that the time between BEGIN/END transaction sequences should be kept short if possible to minimize others from lockout.

At the time this document was prepared there was no information available on how ORACLE implemented or is implementing its journaling and recovery capabilities. However, information is anticipated on these features in the near future.

ORACLE provides two utility routines to backup and restore a data base. EXPORT is used to save all or selected parts (unload) of a data base in a dump file while IMPORT is used to restore all or selected parts (reload) of a data base. EXPORT can dump all tables in a data base or only user-specified tables (including no tables), all or only user-specified views (including no views), all granted privileges or no privileges associated with the tables and views being unloaded or EXPORT may only unload the table/view definitions and their associated privileges. IMPORT allows a user to reload all or none of the tables on the dump file, all or none of the available views, all or none of the available privileges, or to inspect the dump file without reloading anything. A user may indicate that tables are to be added to an existing data base so rows for an existing table will be inserted. If this is not indicated no data will be entered. It is evident that the EXPORT/IMPORT procedure could be used for more than just backup and recovery. It can facilitate the combination of two existing data bases, the respecification of privileges or grants, the dumping of single tables, or the redefinition views. The nature of EXPORT and IMPORT require that the information they process be handled on a record level. The implication is that the saving and restoration of a data base could be as expensive (in computer resources) and time consuming as the loading of the data from its origin if that were a possible alternative. To provide a backup of a large data base in a timely fashion one would probably wish to discard the use of EXPORT and use a VAX file copy utility routine. This would greatly reduce the time required for the backup as well as for the recovery if needed. It would not provide the options available through EXPORT and IMPORT however. In the event that a new ORACLE version uses different internal storage methods EXPORT and IMPORT could be used to unload and reload the data base so the new version could be used.

3.6.2 SEED

Numeric data introduced to SEED has the same restrictions normally imposed on it by the VAX/VMS operating system. An integer number as large as 2,147,483,648 may be input to SEED. Floating point or real numbers are subject to the same loss of accuracy that the VAX/VMS system exerts on all users. This means that a floating point number inserted into the data base may be retrieved with a slight variation in value.

When a user wishes to use GARDEN or the HLI to perform an update (MODIFY) he must first log on to the data base by explicitly naming the sub-schema to be used and identifying his intent to perform an update(s). SEED then attempts to "lock" all areas that are included in references in the sub-schema definition. If a referenced area is already "locked" for another user's update then the log on is rejected and the user must wait until the area is released. If the sub-schema explicitly references records in all the data base's areas or implicitly references all areas through the "COPY ALL" option, the entire data base is locked against other users changing the data base. Other users may log on at any time in a read only mode regardless if area "LOCKOUT" is active. This approach eliminates the possibility of the "deadly embrace" phenomenon but does not exclude the chance that data one person is inspecting has been modified by another user without the first's knowledge.

Several levels of error protection and data base corruption prevention are available or present in SEED. Currently the schema definition identifies one of four modes of journal operation which all users of the data base are subject to. (SEED may implement a deferred mode which will permit users to choose the mode they wish through sub-schemas or at run time but at present all users must use the mode identified in the schema.) The four modes are progressively inclusive, that is, each successively defined mode contains all the features of the preceding mode. The initial mode is "INTERNAL" integrity which guarantees pointer integrity of all chains in the data base. This is always present in SEED. The second mode is "COMMAND LEVEL" integrity which guarantees command completion with rollout or restart. The third mode is "TRANSACTION LEVEL" integrity with roll forward. This permits the application of journalized transactions to a backup copy of the data base so that the updates can be re-applied to the data base. The fourth mode is "TRANSACTION LEVEL" integrity with roll forward and roll back. This adds the ability to retract or erase transactions made to the data base from the current data base contents. Each successive operating mode provides additional capabilities but at increased cost in system performance so one must analyze his needs carefully to strike the proper balance. The user or DBA must use the journaling utility , DBJRNL, to appropriately restore the data base to a desired point after it has been corrupted. The journal file can be inspected to aid in determining how the restoration can best be made. A user may also explicitly identify the start of a logically related sequence of DML commands. He then has the option of identifying the completion of the commands through a "COMMIT" specification or he may roll back the commands completed since the sequence started and thereby erase those commands. He may also generate a checkpoint to the journal log file but care should be taken in a multi-user environment because the checkpoint is for all users. Restoration to the checkpoint would mean other user's inserts, updates and deletes would be erased!!

SEED has two utilities called TROUT and TRIN which may be used to unload all or selected areas of the data base. TROUT and TRIN are part of SEED's SPROUT which is a system for processing transactions into and out of a data base. SPROUT accommodates a variety of data base related transactions and TROUT and TRIN are integral to these functions. Purely for the purpose of backing up and restoring the data base TROUT and TRIN are sufficient but may not be efficient enough for practical day to day use. They are interpretive in nature and, as such, add additional overhead to the cost of dumping and restoring the data. Their use requires an understanding of the data base structure for design and specification of the necessary Transaction Description Language (TDL). Preferable to using TRIN and TROUT is the use of VAX/VMS file utility routines to copy or restore the files comprising the data base. Procedurally this is a simpler task which requires no unique understanding of the data base design and is a much faster process. TROUT and TRIN can be used to unload and reload a data base when a new version of SEED is implemented that employs a different internal management technique.

3.6.3 RIM

During our use of RIM, data inconsistency has not occurred. RIM DBMS provides no capability to recover from failures and/or error. The three VAX files containing the directory data, the data for each table, and the KEY or indexed element pointers should be backed up periodically to disk or tape using the VAX utility COPY.

RIM is a single user DBMS so there is no need to resolve contention in case of multiple accesses to a given piece of information.

3.7 Complimentary Software

Associated with both the ORACLE and SEED DBMSs are an array of software routines that are either required to perform certain DBMS functions or which can be utilized to give more versatility and power to DBMS performance. At this writing both systems are adding to their respective inventory or support software as well as enhancing their current routines. For that reason the discussion that follows cannot be considered as a full-measure of the accessory software available. It is intended that the information presented reflect the knowledge gained about software that was both available during the benchmarking exercises and which time and resources permitted informal testing of.

3.7.1 Complimentary Software of ORACLE

3.7.1.1 Data Base File (DBF) Utility

DBF supports the establishment and mapping of an ORACLE data base. The DBF utility is used to create, initialize, modify, and delete a data base. It also may be used to identify the system data base and to list information about a data base. A user must use the DBF command with the "CREATE" parameter to initially define the data base. When using the "CREATE" command one must identify the new data base's name, an initial file name and its size (a minimum of 1024 blocks of 512 bytes per block is required) and optionally a user-name/password combination. The initial file may be added to through use of the DBF command, "EXTEND", to provide additional data base space. The optional user-name/password specification allows data bases to be secured. If the user-name/password is specified all DBF functions must also specify it when referencing the data base except the SYSTEM DATABASE, LIST, and ENTER functions. The "INIT" and "INITEXTENT" commands initialize existing files as a data base and enter the new data base into the data base directory. The "REMOVE" command removes the data base from the ORACLE data base directory. The "ENTER" command enters a set of initialized data base extent files into the data base directory. A set of files whose data base name has been "REMOVED" may be re-entered using the "ENTER" command. The "MOVE" command allows an extent to be renamed so that a duplicate file may be used in its place. The "REINIT" command flushes all the information in the data base leaving it totally empty.

The command "SYSTEMDB" is required to identify the system data base to ORACLE. The "LIST" command results in information being displayed about the data base of concern or about all the data bases in the directory if desired. The information provided by "LIST" includes the data base name, the user name if it is a secure data base, an indication if the journal option is on or not, and the size and file name of the data base extents. The "LU" command identifies the number of totally unused blocks present in the entire data base. The "M" command provides a hexadecimal output of the bitmaps used by ORACLE to manage space in the data base. At this time no documentation has been made available about either the "LU" or "M" commands.

The DBF utility is required in order to use the ORACLE DBMS. It is generally easy (for a DBMS specialist) to use and is sufficient in its ability to create, modify, and delete data bases. The "LIST" command is helpful for managing a large data base. The "LU" and "M" commands are routines which probably could use enhancing (as well as documenting) to aid DBMS specialists in scrutinizing the data base. Enhancements to include additional statistics on B-tree space consumed, view definition space available, and local dictionary space available would be useful and should be supplied. The use of DBF is not normally a routine that an end user of a data base would be expected to be conversant with. If the end user must create his own data bases, the use of DBF is required and could present an obstacle because of its lack of user-friendliness. Normally the assistance of a DBMS specialist would be required in such a case.

3.7.1.2 Interactive-Application Facility (IAF)

The IAF permits the development of interactive applications for data entry, data retrieval, and update. The IAF application requires a DBMS specialist or designer to define the prescribed application interactively to the Interactive Application Generator (IAG). The designer defines the data base being referenced, the columns and tables of concern, a column's editting criteria, its initial value, a SQL statement to be executed when the field is entered, the fields screen location, and the placement of prompts, instructions, and line drawing characters. The information input to IAG is saved in a response file and may be edited for update to avoid the respecification of all the information interactively again.

Once the definition is complete the application is compiled into an internal format and stored for use by the Interactive Application Processor (IAP).

The terminal operator who wishes to use the application must run the IAP utility and specify the proper image file name created by the IAG. The user may then interactively communicate with the IAP in a manner prescribed by the image file. He may proceed from field to field and from screen to screen during the course of his session until he has completed his assignment. The IAP communicates with ORACLE via the HLI and as such passes SQL statements based on the criteria presented to it by the user and the image file.

The IAF provides a generic capability to define and process interactive applications that communicate with an ORACLE data base without requiring unique software. The approach relieves the terminal operator from having to know anything about the data base. Procedurally he is at the mercy of the designer to produce screens that are easily comprehensible. He must also be aware of the IAP keyboard function codes to proceed from "insert" to "inquiry/update" mode or to "next block" or "previous block". The IAG does not pre-determine the validity of field and table names made to it. Errors of this type would be discovered during the use of the application under IAP which underlines the need for careful checkout prior to the operational use of the design. It also means care should be taken to analyze the effects on all IAF applications whenever a data base redesign is considered or made.

3.7.1.3 Report Writing and Text Formatting Utilities

The Report Writer Utility (RPT) interprets and executes a report program consisting of report writer statements, text formatting commands, and user text. It creates an interim file which may be used as input by the Text Formatting Utility (FMT) to produce a finished report. FMT formats the text based upon embedded commands in its input file and a set of switches specified when it is exectued. Also, FMT may be used in a stand-alone fashion as a general purpose formatter for word processing applications. In that case the input file for FMT is built using a standard text editor rather than by RPT.

During the report generation process, RPT reads a report program created by a standard text editor in a file to be passed as input to RPT. User text and FMT commands in the report program are copied to the interim file but are otherwise ignored by RPT. Report writer statements are interpreted and executed to direct the retrieval of database information and its placement in the interim file.

FMT uses the text in the interim file for titles, column headings, and other descriptive information. The embedded FMT commands are used to control the placement of text and data into a tabular format and to specify spacing, underscoring, margins, and page numbering. FMT does not access any ORACLE data base. To modify a query imbedded in a report requires the entire process be repeated with the new query.

The initial development of a report using FMT and RPT would normally be accomplished by a DBMS specialist. Once the input file is defined and verified any user can initiate the report request to generate the desired output with ease. The definition of the input file requires experience with formatting and knowledge of the data base as well as a set of requirements describing the nature of the report.

3.7.1.4 Unload/Reload Data Base Utility

The Unload/Reload Utility consists of the modules, EXPORT and IMPORT. They are designed to be used in conjunction with each other, that is to say, EXPORT creates a dumpfile which IMPORT can read to rebuild a data base. The EXPORT function permits the dumping of all or only specified data base tables, all or only specified views, all or no GRANT privileges, or only the table/view definitions and GRANT privileges. When specified tables or views are desired EXPORT will prompt for user specification. The names of all tables and views unloaded will be displayed as well as the row count of any unloaded table.

Once a dumpfile has been created IMPORT can be used to access specified portions of it for reloading of the data base or simply for inspection. IMPORT permits a user to specify that all or no tables present on the dumpfile be reloaded, that all or no views present be reloaded, that all or no GRANT privileges be reloaded, that the reload is against an existing data base so tables already present shall have the same dumpfile table's rows added to the existing data or to ignore the dumpfile's data for duplicated table names, or finally to display the name of all tables and views present on the dumpfile. Any records that cannot be processed will be printed and IMPORT will provide a count of the number of rows inserted as well as the number read.

The use of EXPORT/IMPORT for data base backup and recovery is not a wise choice except for relatively small data bases. The use of a VAX file copy utility is far more efficient for this purpose. One occasion when EXPORT/IMPORT would be of use is when a modification to ORACLE results in an internal data base storage change. In this case the data base has to be "EXPORT'ed" using the old version of ORACLE and "IMPORT'ed" using the new version. EXPORT and IMPORT work on a row basis and for that reason are not any faster than an efficient HLI routine that reads or writes rows from or to the data base. They do, however, replace the need for the development of such routines and offer flexibility in the amount and type of information loaded and unloaded. It should be pointed out that the unloading of data from a data base using EXPORT does not remove that data but instead copies it to a sequential file. The format of the dumpfile is not provided by RSI which precludes the opportunity of pre-processing data destined for the data base into a compatible dumpfile format which IMPORT could process for loading into the data base.

3.7.2 Complementary Software for SEED

3.7.2.1 DBINIT

DBINIT is a utility routine which is used to initialize a data base before data can be stored into it. It will also flush existing data if it is run against an area which has been partially or fully loaded. DBINIT sets up the necessary internal SEED pointers and variables in an area so that it will appear empty and be ready to receive data. DBNINIT will permit the user to specify the area(s) which should be initialized if one or more areas do not require it. The use of DBINIT would normally be confined to a DBMS specialist as opposed to an end-user.

3.7.2.2 DBDUMP

DBDUMP is a utility which permits a user to inspect the contents of desired data base pages in both numeric (hexadecimal or decimal) and ASCII. The DBMS specialist can use DBDUMP to display data base pages in order to analyze problems or to empirically study the results of different designs. The provision of a utility like DBDUMP can be of great assistance in the diagnosis of a problem because it facilitates a look at the internal data base structure. It is not anticipated that the end-user would make use of DBDUMP, however.

3.7.2.3 DBSTAT

The DBSTAT utility is used to produce usage reports about the data base. The reports can summarize usage based on user specification for all areas in the data base, selected areas, or selected pages in selected areas. Five kinds of reports are generated by DBSTAT, including: a record instance report, a storage distribution report, a storage utilization report, an all area summary, and page level statistics. (The first three of the above are always provided.) The record instance report provides a count by record type of the number or record occurrences present. The storage distribution report summarizes the number of data base pages which are from 0% to 5%, 5% to 10%, 10% to 15%, ... , 95% to 100% full. Examination of this report shows how well the data is distributed in the data base. It can help determine if data is "bunching up" or if it is evenly distributed. The storage utilization report identifies how a data base page is being used. It identifies the number of bytes and corresponding percentage space associated with page header overhead, record header overhead, set linkage overhead, data item storage, line number placeholders, unrecovered free space, and free space. The report quickly summarizes how the data base space is being consumed and how much space is still available. If the all area summary report is requested the same information as the storage utilization report is provided but for the entire data base area instead of the page level. The page level statistics report lists for each page the percentage of free space, the total number of records on that page, and the total number of records of each record type on the page.

The reports provided under DBSTAT are complementary to the data base. Their use coupled with analysis and some experimentation can aid in the fine tuning of a data base to maximize its effectiveness in meeting user requirements. Note is made that when a "dynamic" data base was used the DBSTAT results were inaccurate but a future version promises to correct the problem.

3.7.2.4 SCDUMP

The SCDUMP utility is used to display the contents of schemas and sub-schemas. It may be invoked interactively or from a FORTRAN program as a subroutine. The contents are provided in an encoded fashion. A user may option to use the text editor to examine his original schema or sub-schema specification as opposed to the encoded output of the SCDUMP output.

3.7.2.5 <u>RECLAIM</u>

The RECLAIM utility removes all deleted records which are present when it is run. When a record is deleted from the data base, it may not be removed from the data base. This implementation was chosen by SEED to minimize the overhead of the delete function. A record will be removed after it is deleted only when all chains associated with the record have been relinked to exclude its occurrence. This may happen during the dynamic navigation through the data base by the initiator of the delete or by another user unintentionally. If it is determined that a large number of deleted but unremoved records exist, the RECLAIM utility can resolve this problem. RECLAIM permits the user to specify the range of pages to be operated upon and RECLAIM reports the number of bytes reclaimed after it scans the specified pages and removes the deleted records. The RECLAIM utility would normally be reserved for use by a DBA or DBMS specialist when it was determined that unremoved records were becoming detrimental to system performance. No figures are available to document how long this function requires but obviously it is a factor of the size of the data base and the number of pointers present.

3.7.2.6 <u>BLOOM</u>

BLOOM is a routine which interfaces with a SEED data base to produce user defined reports. BLOOM eliminates the need for user provided DML statements by locating target records and determining the access path used for data retrieval. The first step in using BLOOM is to use the text editor to create a file which contains the report definition language (RDL) which defines the report format and contents. A great deal of flexibility is provided in the format definition which offers a wide latitude in the report layout. Once completed the RDL is input to the Report Definition Processor (RDP) which compiles it to produce a report format called an FMT file which is later accessed by the BLOOM processor to produce a report. RDP will analyze the report and determine if access paths and target records can be derived from the data base using the sub-schema provided. If not, error indications are given to inform the user of the problem.

Once the FMT file is successfully produced it may be used repetitively by the BLOOM processor to produce the report as needed without further user alteration unless the report format or contents requires alteration. When the BLOOM processor is used to produce a report it interfaces with SEED and creates a report data file (RDF) and a report auxiliary data file (RAD) which hold the unformatted results of the report. The user can specify the re-use of the RDF and RAD files to regenerate a report that has already been created without needing to access the data base again. He may produce a report that contains only summary information and he may also select to output the report to a terminal or to a disk file for future printing. Additionally the user may specify selection criteria to reduce the amount of information included in his report to what is desired. Selection criteria can refer to items in the data base, defined variables from the RDL specification or summaries. To modify a query defined in the report requires the entire process be repeated with the corrected query.

The use of BLOOM has been limited to date but one problem has arisen with the sorting of real numbers. SEED apparently uses the VAX sort capability and this currently causes erroneous results when used with SEED managed real numbers. Otherwise there were no significant errors to be noted in the use of BLOOM. As might be expected, a DBMS specialist is recommended for the job of creating the RDL and producing a verified FMT file which end users could then run the BLOOM processor against to produce reports.

3.7.2.7 SPROUT

SPROUT is a system for processing "transactions" into and out of a data base. It can be used for outputting data to external files and inputting data from external files among other things. The SPROUT system consists of four processors: the Transaction Definition Processor (TDP) which defines record (transaction) formats of external files into a transaction library; the Transaction Library Dump (TLDUMP) which displays format definitions; the Transaction Input Processor (TRIN) which creates or modifies a data base from an external transaction file; and the Transaction Output Processor (TROUT) which creates a transaction file from a SEED data base.

TDP reads and checks transaction definitions and creates a Transaction Library using an input file of Transaction Definition Language (TDL) and a sub-schema. The TDL permits the specification of up to 50 transaction definitions for defining the record formats to be input or output to/from the data base. If any TDL errors are encountered, TDP identifies them including data name and field-length errors to the user. Once properly compiled into the Transaction Library TRIN and TROUT can be used accordingly. TLDUMP can be used to print a formatted directory of any Transaction Library that has been created by TDP.

The use of TROUT and TRIN for system backup and recovery is not recommended. The use of VAX file copy utility is advised for the sake of efficiency. TROUT and TRIN are general purpose transaction processors that interpretively decode a transaction from the Library definition and operate accordingly. This implies a significant amount of overhead that is not necessary. For a large data base, transaction processing of all data base records would be prohibitive on a regular basis. To accommodate a data base design change or an internal SEED Modification one might consider their use but should not overlook the use of FORTRAN or COBOL interfacing with the data base through the HLI. That approach if done properly will normally reduce computer resources consumed in the backup and recovery steps. The comprehension of the data base structure required for either approach is probably equivalent so this is not a factor. In either case it is advisable that a DBMS specialist be employed to perform the work as this is not a job safely left to an end-user.

3.7.3 Complementary Software of RIM

All of the capabilities which RIM provides are accessed through the on-line terminal interface or the programming language interface; there are no additional utilities.

4.0 SUMMARY

4.1 Load Rates

The load rates of the systems varied in different degrees as the one million Climate Data Access System (CDAS) records containing the FGGE/LIMS information was input to the data bases. As noted, the ORACLE data base initially began the loading process with load rates of about 7.4 CDAS records per second and was extremely uniform in performance. It demonstrated a rather small degradation in performance as it eventually dropped to a level of 6.2 records per second at about the one million record (CDAS) level. RIM which did not index the time field to the Profile record and therefore was subject to less processing overhead was loading at over 23 CDAS records per second at the 100,000 level. (The initial attempts to load RIM with the Profile Time field as an index resulted in rates which were unacceptable for continued testing and, for this reason, the index was excluded from the design.) The RIM system load rates demonstrated more fluctuation and degradation than the ORACLE rates. Perhaps the dynamic acquisition of file space used by RIM periodically induces an overhead which is responsible for the fluctuations. The degradation drops the load rate to approximately 6.7 CDAS records per second at the one million record data base level. The SEED load rates demonstrated even more fluctuation and degradation than RIM's. Initially, the SEED load rates were over 45 CDAS records per second but by the one million data base level the rate was reduced to below 5 records per second. The fluctuations that are so evident during the SEED loading are attributed mainly to the data location technique used by SEED called "hashing." SEED provides a default algorithm which operates on "key fields" to determine a location for the data record to reside. If the location determined is already full an overflow occurs adding overhead to the locating process. This periodic overhead is thought to be the cause of the fluctuations observed during the SEED loading.

The cumulative time required to load the one million plus CDAS records into the three systems is depicted in Figure 4.1. The RIM and SEED systems appear to have required almost the same amount of time, approximately 33 hours, to load the records. ORACLE has consumed approximately 42 hours to perform the same amount of loading.

From these results several observations may be made about the loading capabilities of the systems. It appears that ORACLE is much less sensitive to data base size than either RIM or SEED when comparing the load rates. For a data base of a smaller size SEED appears to have an advantage in load performance but above the 700,000 CDAS record level in this application it is clear that ORACLE outperforms SEED. One cannot directly compare the RIM results because of the omitted index field but it would appear that its degradation in load performance indicates that ORACLE would begin to outperform it even with the omitted index field at a data base size of slightly more than one million records.

Some reservations about the test results should be stated as well. Other IED test applications have demonstrated load rates with significantly different results. This underscores the application dependence of the results which one must consider when evaluating them. Additional testing is needed to evaluate other variables which may effect the load performance such as the number of fields in a record, the number of keys or indexed values, the nature of an indexed fields values (i.e. character vs. integer, length, duplicity), as well as the number of different records. The SEED DBMS offers a number of options in the data base design that also influence load rates which are not available in ORACLE or RIM. These options represent another set of variables which could be subjected to analysis to aid in the evaluation of load performance. In some applications a particular option could be utilized to improve performance while in others its use may be a hindrance.

The percentage of CPU time consumed by each system for each second of elapsed wall clock time is worth noting as well. ORACLE consumed

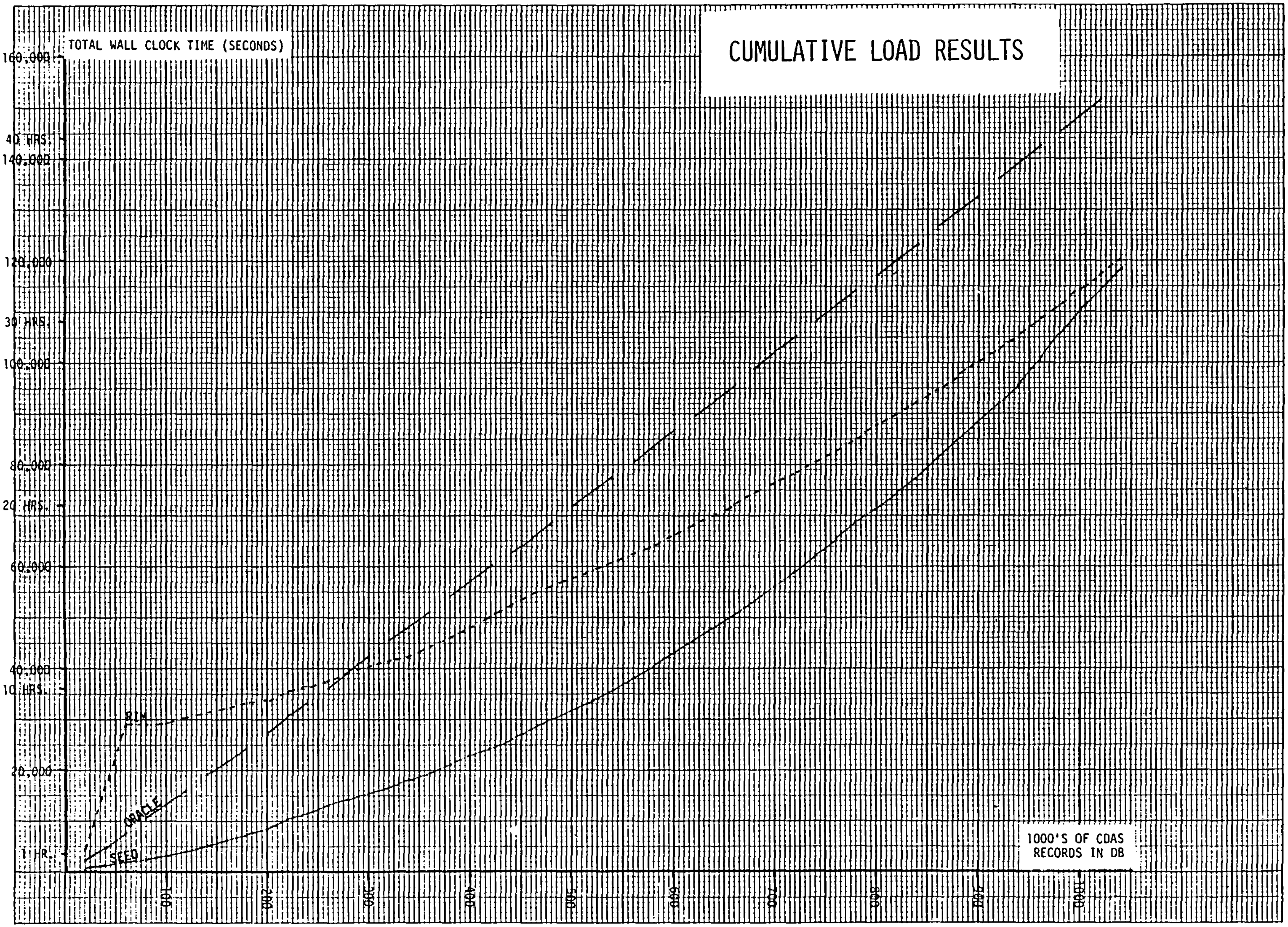

CUMULATIVE LOAD RESULTS
TOTAL WALL CLOCK TIME (SECONDS)
160,000
40 HRS.
140,000
120,000
30 HRS.
100,000
80,000
20 HRS.
60,000
40,000
10 HRS.
20,000
1 HR.
RIM
ORACLE
SEED
100
200
300
400
500
600
700
800
900
1000
1000'S OF CDAS RECORDS IN DB

Figure 4.1

about 75% of the available processing time while SEED used about 50% and RIM used from 34% to 24% (RIM used less as the data base grew). The implication is clear that in an environment that must share resources with other users ORACLE load performance would suffer a more drastic reduction than SEED's which would suffer more than RIM's. The impact in performance may be significant and further testing of the systems should include an investigation of the effect of contention for the host computer's resources.

An examination of the disk memory utilization of each data base containing approximately 60,000 CDAS Profile records and 980,000 Entry records was made. The SEED data base reported via its utility software that less than 20 million bytes of storage has been consumed of the space originally allocated for it. The files that contain the RIM data base consume about 40 million bytes, but recall it does not contain an index for the Profile time value. ORACLE consumed about 87 million bytes but of that 43 million is thought to be overhead attributed to the current 64 byte management scheme employed. The promised 2 byte approach would improve this drastically but it will increase the amount of bit map space required by a factor of 32 as well incrementing the processing overhead some.

4.2 Query Testing

The TI results for all three systems under scrutiny show adequate response. An adequate response in an interactive environment implies that a momentary pause is acceptable to the user.

The results demonstrated that queries made with conditional clauses referencing indexed fields were met with no more than momentary pauses and are, therefore, considered adequate and effective. Access to non-indexed data or for summary information requiring the sequential searching of large portions of data are not recommended as regular functions in an interactive mode. If this happens a data base design change should be considered to expedite the processing such as the keying of additional fields or the maintenance of on-line summary information dynamically.

The HLI results are worth examining more closely because of the potential for a small difference in response becoming significant due to repetitive interfaces with the data base. The results show that SEED was generally the fastest to respond to queries with conditional clauses specifying an indexed field. RIM was next and ORACLE was slowest. To locate a particular indexed value associated with the Profile record information from among as many as 26,000 possible choices, SEED requires about .1 seconds. From among 60,000 possible choices SEED required about .3 seconds. Corresponding response times for RIM are .3 and .5 seconds and ORACLE consistenty requires about .5 seconds at all levels. To locate a particular indexed value associated with an Entry record from among as many as 413,000 choices SEED requires about .3 seconds and from 980,000 choices about .75 seconds. Corresponding times for RIM are .5 and .75 seconds while ORACLE took between about 1.1 and 1.3 seconds for all levels. The differences between the systems are small but the cumulative effect could be significant if an application required a high volume of responses in a relatively short period of time.

HLI test which sequentially accessed large portions of the data base were done to determine the cost of accessing non-indexed fields or to obtain summary information from the data base such as when making periodic reports. At the largest level the data bases contained 980,000 Entry records. To access all of the information via the HLI, RIM required less than 17 minutes. In contrast SEED requires over 63 minutes and ORACLE requires over 222 minutes.

4.3 Qualitative Aspects

The ORACLE and RIM systems were found to require a great deal less comprehension of data base theory than the SEED system to design and implement a data base application. The two dimentional tables are more readily understood than the network approach using member-owner sets. The relational systems are also more permissive to modification of the original design. A variety of alterations can be made including the addition of space to the data base, the addition of tables in the data base, and the addition of fields in a table without unloading and reloading the data base. SEED's schema approach is much more rigid and will not readily support modifications to the original design.

The ORACLE and RIM systems do not offer a wide variety of options that permit a data base application to be tuned for special needs. SEED provides a variety of permissible options in the schema specification to facilitate special needs or aspects of the application. For example, one may select from three choices of "location mode" to designate the manner in which a record type may be loaded into the data base. A hashing technique may be specified to facilitate direct access, a "via" technique to maintain physical proximity to an owner record (reducing disk I/O's) may be specified, or a "direct" technqiue may be used to enable sequential access or loading in a more efficient manner. This type of latitude is not available in either ORACLE or RIM. A more naive user must be cautioned to examine his data base needs carefully before selecting the techniques for his schema specification because an unwise choice may degrade the actual performance he desires to emphasize.

Both ORACLE and SEED have an assortment of complimentary software including utility routines necessary for creating and maintaining a data base as well as peripheral modules such as report writers that increase the overall DBMS usefulness. This software adds to the flexibility and power of the two systems but also underscores the need for cognizant data base personnel to aid in the proper use of these routines. RIM, at this writing, does not possess the peripheral software available in either SEED or ORACLE. Its biggest drawback, however, appears to be the limitation of only one user in version 4.0. Without enabling multiple users RIM's suitability is highly suspect for most data base applications. It is unknown at this time if plans exist to adapt RIM for multiple user access. If so, the revised system should undergo more testing because the nature of the changes required would most likely modify response performance.

4.4 Conclusions and Recommendations

The major goals of the testing were accomplished but the results of the tests were somewhat inconclusive. It is clear that none of the systems meet all the requirements originally stated for the IDBMS (now the PMS). Because of RIM's limitation for supporting only a single user it can probably be eliminated from consideration as a candidate. The other two systems are still worthwile candidates for use as a "nucleus" for the PMS data base management software. Either can be surrounded with the cutomized software required for the unique needs of the PMS.

The results of the testing provide some basis for estimating performance of the systems. As already mentioned the load rates derived in the tests demonstrate that a comparison of the systems performance must be done with respect to data base size since different rates exist at initial levels but degradations exist which eventually change the order of performance. Other factors including contention for CPU resources by other users which is likely in the PMS environment have not been included in this study. Because of the different percentage of CPU time required by ORACLE and SEED during loading, approximately 75% vs. 50%, contention could more adversely effect one system than the other. Further, the application dependence of the test results should be underlined. The variables associated with the application including number of record types, size of records, numbers of indexed fields among others will impact the load performance. Informal testing of other IED applications have produced significantly different load rates initially. Consequently, a recommendation is made that additional testing be performed which uses a pseudo PMS design to minimize the application variability if conclusions are to be firmly drawn regarding the performance of the DBMSs in the PMS environment.

results. If the VAX computer was burdened with heavy use the DBMS's could not be expected to perform as well. The DBMS's have been periodically updated with new software releases. They may continue to be updated in the future as they mature. The changes made to the software can have significant effects on performance. For example, the proposed 2-byte memory management capability in ORACLE and the pointer array capability in SEED could greatly enhance each system respectively. Additional testing should be considered for these systems after such changes to re-evaluate performance. Lastly, the reader should recall that the evaluation and testing made has been relative to the typical needs of a NASA data base application. These needs are not necessarily oriented the way those of an average commercial data base might be. The typical NASA data base is presumed to consist of large amounts of scientific data that once loaded will remain somewhat static.

APPENDIX I
QUERY PERFORMANCE RESULTS

APPENDIX I
PART A
BENCHMARKING OPERATIONS

1. Using an indexed field, locate a specific Profile related value.

2. Sequentially access all the Profile records.

3. Using an indexed field, locate a specific Entry related value.

4. Sequentially access all the Entry records.

5. Establish the cost of compound selection criteria
   A.) With a single indexed value as selection criteria (control)
   B.) With two indexed values as selection criteria
   C.) With an indexed value and a non-indexed value as selection criteria

6. Measure sort capability
   A.) Unsorted (control)
   B.) Sorted

7. Incremental addition and deletion of a Profile value
   A.) Insertion
   B.) Deletion

8. Incremental addition and deletion of an Entry value
   A.) Insertion
   B.) Deletion

9. Open and close data base without intermediate operations

APPENDIX I
PART B
TABLES OF BENCHMARKING RESULTS

Note should be made that at the 52,000, 439,000, and 1,040,000 row data base levels version 2.3 of ORACLE was used while at the 99,000 and 189,000 levels the measurements of the ORAAAA task are approximated. At the 52,000, 189,000 and 1,040,000 the ORAAAA times are derived from the system accounting log but at the other two levels the times, I/O, and page faults had to be derived from the latest update of the DEC display monitor prior to the conclusion of the primary task.

Also, of importance is the fact that all terminal interface results (UFI, GARDEN, HARVEST, and TI) include the cost of opening and closing the data base in their measurements. One might attempt to more accurately estimate response and overhead by determining a net time by subtracting out the appropriate query 9 results which simply opened and closed the data base without intermediate steps.

The tables that follow contain the results of the Benchmark tests at each level of data base size. The left hand column specifies the data base size. For consistency, the size stated is related to the number of CDAS records loaded, not necessarily the number of data base records. The measurements provided are for four different resources identified in the second column as: clock time, CPU time, Direct I/O, and page faults. Clock time is the total number of seconds required from submission to completion of a function. CPU time is the computer processing time consumed while performing a function. Direct I/O's are the number of disk read and writes issued by the software. Page faults are the number of times the software addresses an instruction or location not currently in main memory thus requiring the virtual operating system to swap main memory for the desired page on the disk. Each of the three systems have separate columns.

ORACLE's column is subdivided into an HLI and UFI (interactive interface) halves each with an ORAAAA element which corresponds to the detached process for the test function. SEED is subdivided into HLI, GARDEN, and HARVEST (BLOOM is used for 6A and 6B) where appropriate since some functions were not tested using all interfaces. RIM was subdivided into HLI and TERM. INTER. only. Neither SEED nor RIM had a detached process to keep track of.

All TI results (UFI, GARDEN, HARVEST, TERM. INTER) contain the added overhead of opening and closing the data base as well as the cost of performing the desired test function. All HLI figures are net results and do not include the overhead. The ORAAAA column for ORACLE's HLI results is blank because measurements were not obtainable for this detached process on a test function basis. It may be presumed that the equivalent ORAAAA results for UFI approximate those for the HLI tests with results from Query #9 subtracted out first

Note should also be made that each level of data base size is made up of a proportionate amount of FGGE/LIMS profile information and entry information. The ratio of the two record types averaged about 1 Profile record to 15.85 Entry records. As a result queries that access Profile related information are searching a much smaller amount of data than those which access Entry related data. An estimate of the number of Profile and Entry records at each tested level of data base size is:

Data Base Size	Profile Records	Entry Records
52,000	3,100	48,900
99,000	5,900	93,100
189,000	11,200	177,800
439,000	26,100	412,900
1,040,000	61,700	978,300

The above figures are expressed in terms of records from the input source (FGGE/LIMS data tapes) and does not necessarily reflect data base records.

BENCHMARKING OPERATION RESULTS #1

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED			RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	GARDEN	HARVEST	HLI	TERM. INTER
52,000	Clock Time (SEC)	.40(.66)*	--	10.01	8.0	.07	2.54	3.31	96.11	109.11
	CPU Time (SEC)	.02(.05)	--	.60	2.63	.02	1.17	1.65	53.42	58.58
	Direct I/O	0 (0)	--	--	37	1	6	6	1712	1792
	Page Faults	0 (19)	--	--	718	11	606	831	752	1999
99,000	CLK Tim	.62*	--	7.42*	7.42*	.06	2.39	2.90	.25	3.17
	CPU Time	.03	--	.52	2.31	.02	1.14	1.52	.12	1.72
	DIR I/O	0	--	--	29	1	6	6	4	83
	PG FLTS	22	--	--	674	11	621	776	24	860
189,000	CLK Time	.48*	--	7.47*	7.47*	.11	1.99	2.82	.32	3.08
	CPU Time	.04	--	.61	1.97	.03	1.07	1.39	.17	1.79
	Dir I/O	0	--	--	22	3	8	9	4	83
	PG. Flts	0	--	--	590	11	573	781	24	874
439,000	CLK Time	.44	--	6.91	6.0	.07	2.27	2.82	.29	5.91
	CPU Time	.02	--	.56	2.50	.02	1.27	1.44	.14	2.38
	Dir I/O	0	--	--	37	2	7	8	4	72
	PG Flts	0	--	--	718	11	597	781	20	858
1,040,000	CLK Time	.52	--	6.76	6.0	.54	3.08	13.36	.52	5.32
	CPU Time	.02	--	.59	2.73	.18	1.32	1.75	.13	1.7
	Dir I/O	0	--	--	38	22	27	28	5	79
	PG FLTS	0	--	--	729	11	631	775	25	879

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #2

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED		RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	HARVEST	HLI	TERM. INTER
52,000	Clock Time (SEC)	103.4 (108.43)*	--	109.03	108.0	106.33	140.16	29.37	36.52
	CPU Time (SEC)	.04 (.04)	--	.58	55.78	31.86	52.67	18.06	21.62
	Direct I/O	3 (3)	--	--	2684	3120	3126	318	397
	Page Faults	0 (1)	--	--	716	1513	3391	69	952
99,000	CLK Time	209.93*	--	214.88*	214.88*	216.67	283.14	57.84	53.81
	CPU Time	.03	--	.67	101.51	60.42	99.20	34.69	36.96
	DIR I/O	6	--	--	5451	5893	5899	601	680
	PG FLTS	1	--	--	676	1592	6621	134	1534
189,000	CLK Time	422.92*	--	283.04*	283.04*	295.48	439.64	92.05	101.64
	CPU Time	.03	--	.63	191.20	102.15	232.59	63.51	68.55
	Dir I/O	14	--	--	10256	8959	8965	1134	1213
	PG. Flts	1	--	--	689	3574	9222	117	2244
439,000	CLK Time	892.38	--	902.87	902.0	660.78	1009.43	214.04	223.77
	CPU Time	.04	--	.65	452.92	228.12	531.62	149.01	160.10
	Dir I/O	29	--	--	23245	18865	18871	2661	2729
	PG Flts	0	--	--	717	6564	17712	142	6106
1,040,000	CLK Time	2132.78	--	2141.38	2140.0	1413.04	2236.61	546.22	560.18
	CPU Time	.05	--	.71	1065.48	510.02	1254.39	350.35	375.01
	Dir I/O	71	--	--	55477	40178	40183	6297	6371
	PG FLTS	0	--	--	729	13468	33628	119	17036

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #3

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED			RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	GARDEN	HARVEST	HLI	TERM. INTER
52,000	Clock Time (SEC)	1.15(1.24)*	--	8.34	7	.32	3.10	3.64	.55	7.68
	CPU Time (SEC)	.17(.12)	--	.83	2.94	.16	1.67	1.88	.26	3.29
	Direct I/O	0 (0)	--	--	39	4	8	8	10	85
	Page Faults	0 (2)	--	--	719	64	649	883	31	889
99,000	CLK Time	1.15*	--	11.51*	11.51*	.22	2.90	3.24	.52	3.83
	CPU Time	.17	--	3.21	3.02	.14	1.55	1.92	.24	1.93
	DIR I/O	0	--	--	42	3	7	7	8	85
	PG FLTS	1	--	--	677	63	600	892	30	872
189,000	CLK Time	1.36*	--	11.84*	11.84*	.31	2.90	2.96	.49	4.35
	CPU Time	.19	--	3.61	3.05	.19	1.61	1.86	.28	1.89
	Dir I/O	0	--	--	36	5	0	10	9	85
	PG. Flts	7	--	--	689	64	642	870	32	877
439,000	CLK Time	1.08	--	8.46	7.0	.33	3.19	3.73	.45	5.98
	CPU Time	.19	--	.83	3.09	.19	1.70	1.99	.24	2.86
	Dir I/O	0	--	--	39	5	10	10	9	74
	PG Flts	0	--	--	719	63	650	854	30	842
1,040,000	CLK Time	1.33	--	8.53	7.0	.74	3.50	3.88	.73	5.45
	CPU Time	.16	--	.79	3.19	.33	1.85	2.09	.26	1.87
	Dir I/O	0	--	--	40	24	28	28	9	82
	PG FLTS	0	--	--	728	64	587	886	23	878

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #4

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED		RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	HARVEST	HLI	TERM. INTER
52,000	Clock Time (SEC)	680.49(926.27)*	--	694.14	693	249.09	548.94	67.83	78.51
	CPU Time (SEC)	.06(.12)	--	.60	598.6	136.74	399.35	53.36	57.75
	Direct I/O	22 (0)	--	--	1,568	3,492	3,498	319	397
	Page Faults	0 (2)	--	--	716	3,191	9,794	221	1,635
99,000	CLK Time	1,891.96*	--	1,851.21*	1,851.21*	521.23	1,107.78	114.32	124.72
	CPU Time	.08	--	1.56	1,413.77	259.87	778.39	96.62	101.68
	DIR I/O	59	--	--	15,659	7005	7,011	603	681
	PG FLTS	13	--	--	679	4,885	17,423	356	2,182
189,000	CLK Time	4,048.75*	--	3,366.34*	3,366.34*	754.14	2,971.15	221.88	140.22
	CPU Time	.38	--	.70	2,427.02	483.47	2,507.12	172.69	100.95
	Dir I/O	123	--	--	29,251	10,845	10,850	1,134	1,213
	PG. Flts	9	--	--	285	8,786	44,545	3,059	965
439,000	CLK Time	5,667.79	--	5,610.83	5,609.0	1,754.56	7,476.69	451.61	326.28
	CPU Time	.06	--	.63	4,906.57	1,142.36	5,836.77	356.57	235.84
	Dir I/O	188	--	--	12,903	22,826	22,832	2,661	2,729
	PG Flts	0	--	--	715	19,686	92,461	7,349	1,045
1,040,000	CLK Time	13,350.21	--	OMITTED		3,811.96	15,703.78	1,000.81	767.88
	CPU Time	.15	--			2,603.52	13,427.67	835.23	557.03
	Dir I/O	444	--			48,258	48,263	6,297	6,371
	PG FLTS	0	--			42,314	221,350	18,357	1,072

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #5A

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED		RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	HARVEST	HLI	TERM. INTER
52,000	Clock Time (SEC)	.39(.57)*	--	6.66	5.0	.12	3.50	.27	7.41
	CPU Time (SEC)	.02(.04)	--	.66	2.47	.04	1.76	.11	3.10
	Direct I/O	0 (0)	--	--	36	2	8	4	83
	Page Faults	0 (1)	--	--	717	28	924	12	864
99,000	CLK Time	.73*	--	12.66 *	12.66*	.15	3.26	.35	4.34
	CPU Time	.04	--	.62	2.73	.04	1.67	.19	1.86
	DIR I/O	0	--	--	53	3	9	6	86
	PG FLTS	0	--	--	690	30	845	13	866
189,000	CLK Time	.8*	--	7.12*	7.12*	.18	4.47	.63	2.79
	CPU Time	.06	--	.61	1.91	.03	2.81	.31	1.53
	Dir I/O	0	--	--	22	8	88	4	10
	PG. Flts	0	--	--	592	15	867	2	758
439,000	CLK Time	.92	--	7.39	6.0	.50	4.06	1.87	7.11
	CPU Time	.04	--	.57	2.73	.10	1.73	.82	3.13
	Dir I/O	0	--	--	46	11	17	20	89
	PG Flts	0	--	--	716	33	784	21	854
1,040,000	CLK Time	1.79	--	8.51	7.0	1.24	4.57	6.96	11.05
	CPU Time	.05	--	.58	3.19	.25	2.10	2.56	3.41
	Dir I/O	0	--	--	65	30	35	58	133
	PG FLTS	0	--	--	727	33	768	41	878

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #5B

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED		RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	HARVEST	HLI	TERM. INTER
52,000	Clock Time (SEC)	.46(.51)*	--	6.71	6.0	.23	3.52	.25	7.25
	CPU Time (SEC)	.06(.02)	--	.60	2.7	.07	1.77	.09	3.07
	Direct I/O	0 (0)	--	--	36	3	9	4	83
	Page Faults	0 (0)	--	--	719	51	881	16	872
99,000	CLK Time	.63*	--	7.50*	7.50*	.31	3.52	.39	4.00
	CPU Time	.04	--	.58	2.52	.07	1.77	.26	2.94
	DIR I/O	0	--	--	39	6	9	6	86
	PG FLTS	0	--	--	677	25	881	18	905
189,000	CLK Time	.63*	--	7.30*	7.30*	.43	2.89	.76	5.89
	CPU Time	.06	--	.58	2.78	.11	1.57	.45	2.09
	Dir I/O	0	--	--	33	8	9	11	91
	PG. Flts	0	--	--	690	51	764	26	850
439,000	CLK Time	.66	--	6.65	6.0	.92	3.05	2.33	8.17
	CPU Time	.08	--	.62	2.66	.22	1.58	1.12	3.61
	Dir I/O	0	--	--	36	20	9	29	98
	PG Flts	0	--	--	719	51	764	41	854
1,040,000	CLK Time	.71	--	7.60	6.0	4.00	3.31	12.27	21.01
	CPU Time	.03	--	.68	2.86	.55	1.67	3.63	5.24
	Dir I/O	0	--	--	40	58	16	114	190
	PG FLTS	0	--	--	728	51	734	46	879

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #5C

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED		RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	HARVEST	HLI	TERM. INTER
50,000	Clock Time (SEC)	.51(.37)*	--	6.57	5.0	.20	--	.28	7.50
	CPU Time (SEC)	.04(.04)	--	.56	2.67	.06	--	.11	3.06
	Direct I/O	0 (0)	--	--	36	3	--	4	81
	Page Faults	0 (0)	--	--	718	51	--	18	678
99,000	CLK Time	.34*	--	7.35*	7.35*	.25	3.15	.31	3.93
	CPU Time	.05	--	.63	2.72	.11	1.83	.17	1.31
	DIR I/O	0	--	--	52	4	10	6	86
	PG FLTS	0	--	--	688	125	1,119	22	866
189,000	CLK Time	.52*	--	6.25*	6.25*	.37	3.37	.55	4.61
	CPU Time	.03	--	.58	2.01	.12	2.02	.34	1.77
	Dir I/O	0	--	--	22	6	12	8	88
	PG. Flts	0	--	--	578	125	1,262	37	861
439,000	CLK Time	1.00	--	7.34	6.0	.69	4.16	2.33	7.97
	CPU Time	.04	--	.60	2.93	.23	2.21	1.09	3.48
	Dir I/O	0	--	--	46	13	19	28	97
	PG Flts	0	--	--	718	125	1,381	42	874
1,040,000	CLK Time	1.86	--	8.41	7.0	1.66	8.22	8.38	13.40
	CPU Time	.08	--	.65	3.36	.56	3.24	2.92	4.86
	Dir I/O	0	--	--	65	34	39	39	166
	PG FLTS	0	--	--	728	200	1,703	1,703	880

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #6A

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED	RIM	
		HLI	ORAAAA	UFI	ORAAAA	BLOOM	HLI	TERM. INTER
52,000	Clock Time (SEC)	81.08	--	181.17	180.0	1774.03	56.14	78.80
	CPU Time (SEC)	16.15	--	33.76	54.38	1540.28	37.28	56.41
	Direct I/O	0	--	--	435	5,627	721	892
	Page Faults	0	--	--	719	35,356	54	1,031

BENCHMARKING OPERATION RESULTS #6B

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED	RIM	
		HLI	ORAAAA	UFI	ORAAAA	BLOOM	HLI	TERM. INTER
52,000	Clock Time (SEC)	93.11	--	287.99	287.0	1732.63	64.96	85.66
	CPU Time (SEC)	16.20	--	34.39	124.83	1549.81	46.53	66.07
	Direct I/O	0	--	--	1688	5,631	780	949
	Page Faults	0	--	--	724	38,487	593	1,511

BENCHMARKING OPERATION RESULTS #7A

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED		RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	GARDEN	HLI	TERM. INTER
52,000	Clock Time (SEC)	.93(1.47)*	--	7.30	6.0	1.09	3.18	15.59	--
	CPU Time (SEC)	.02(.04)	--	.64	2.81	.20	1.62	7.82	--
	Direct I/O	0 (0)	--	--	49	21	17	549	--
	Page Faults	0 (19)	--	--	718	75	1,091	58	--
99,000	CLK Time	1.14*	--	8.28*	8.28*	1.13	3.74	.43	--
	CPU Time	.02	--	.57	.45	.18	1.48	.21	--
	DIR I/O	0	--	--	3	21	21	12	--
	PG FLTS	0	--	--	417	73	790	24	--
189,000	CLK Time	1.31*	--	8.57*	8.57*	.84	3.03	.53	--
	CPU Time	.03	--	1.45	2.48	.19	1.48	.24	--
	Dir I/O	0	--	--	29	21	18	13	--
	PG. Flts	0	--	--	685	73	759	27	--
439,000	CLK Time	1.97	--	11.34	9.0	.88	3.30	.81	--
	CPU Time	.03	--	.56	3.02	.17	1.53	.28	--
	Dir I/O	0	--	--	48	22	23	22	--
	PG Flts	0	--	--	719	73	788	26	--
1,040,000	CLK Time	1.33	--	7.97	6.0	.89	6.53	3.8	--
	CPU Time	0	--	.51	2.97	.21	2.68	.6	--
	Dir I/O	0	--	--	50	25	153	52	--
	PG FLTS	0	--	--	728	70	823	26	--

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #7B

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED		RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	GARDEN	HLI	TERM. INTER
52,000	Clock Time (SEC)	.79(.62)*	--	6.96	6.0	.19	1.36	--	7.29
	CPU Time (SEC)	.02(0)	--	.58	2.68	.07	1.21	--	3.10
	Direct I/O	0 (0)	--	--	46	4	9	--	91
	Page Faults	0 (0)	--	--	711	47	595	--	909
99,000	CLK Time	.31*	--	8.43*	8.13*	.18	2.19	--	3.70
	CPU Time	0	--	.58	.82	.04	1.13	--	1.78
	DIR I/O	0	--	--	9	4	9	--	90
	PG FLTS	0	--	--	463	47	595	--	902
189,000	CLK Time	.33*	--	7.59*	7.59*	.16	2.24	--	6.13
	CPU Time	.01	--	.53	.60	.07	1.13	--	1.87
	Dir I/O	0	--	--	9	4	9	--	90
	PG. Flts	0	--	--	450	47	591	--	873
439,000	CLK Time	1.31	--	7.23	6.0	.13	2.29	--	6.01
	CPU Time	.03	--	.55	2.64	.02	1.12	--	2.60
	Dir I/O	0	--	--	46	4	9	--	79
	PG Flts	0	--	--	715	12	777	--	858
1,040,000	CLK Time	1.03	--	7.49	6.0	.24	3.55	--	5.58
	CPU Time	.02	--	.53	2.74	.09	1.32	--	1.74
	Dir I/O	0	--	--	47	6	34	--	86
	PG FLTS	0	--	--	726	47	679	--	906

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #8A

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED		RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	GARDEN	HLI	TERM. INTER
52,000	Clock Time (SEC)	.85(.71)*	--	10.72	8.0	1.4	4.33	.22	--
	CPU Time (SEC)	.01(.01)	--	.63	2.9	.09	1.39	.10	--
	Direct I/O	0 (0)	--	--	39	2	10	4	--
	Page Faults	0 (0)	--	--	718	67	672	14	--
99,000	CLK Time	.69*	--	7.91*	7.91*	.16	2.44	.19	--
	CPU Time	0	--	.52	2.69	.07	1.32	.11	--
	DIR I/O	0	--	--	39	2	8	4	--
	PG FLTS	0	--	--	677	67	612	14	--
189,000	CLK Time	.83*	--	7.97*	7.97*	.11	2.46	.28	--
	CPU Time	.02	--	.63	7.48	.06	1.35	.10	--
	Dir I/O	0	--	--	29	2	9	4	--
	PG. Flts	0	--	--	685	67	729	15	--
439,000	CLK Time	.82	--	7.27	6.0	.14	2.61	.22	--
	CPU Time	.02	--	.53	2.93	.09	1.32	.12	--
	Dir I/O	0	--	--	39	2	9	4	--
	PG Flts	0	--	--	719	67	608	15	--
1,040,000	CLK Time	.99	--	7.3	6.0	2.08	3.70	.49	--
	CPU Time	.02	--	.56	3.03	.92	1.53	.14	--
	Dir I/O	0	--	--	42	108	31	5	--
	PG FLTS	0	--	--	729	67	656	15	--

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #8B

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE				SEED		RIM	
		HLI	ORAAAA	UFI	ORAAAA	HLI	GARDEN	HLI	TERM. INTER
52,000	Clock Time (SEC)	.59(.26)*	--	6.98	6.0	.15	2.76	--	8.06
	CPU Time (SEC)	.02(.02)	--	.60	2.75	.06	1.17	--	3.17
	Direct I/O	0 (0)	--	--	38	2	8	--	84
	Page Faults	0 (0)	--	--	711	61	622	--	880
99,000	CLK Time	.26*	--	7.40*	7.40*	.13	2.53	--	3.67
	CPU Time	.01	--	.58	.34	.07	1.18	--	1.71
	DIR I/O	0	--	--	3	2	8	--	84
	PG FLTS	0	--	--	282	61	580	--	872
189,000	CLK Time	.4*	-	7.68*	7.68*	.10	2.39	--	3.95
	CPU Time	.01	--	.53	1.41	.05	1.13	--	1.75
	Dir I/O	0	--	--	12	2	7	--	84
	PG. Flts	0	--	--	546	61	593	--	866
439,000	CLK Time	.63	--	7.11	6.0	.14	2.25	--	5.74
	CPU Time	.01	--	.54	2.61	.08	1.14	--	2.40
	Dir I/O	0	--	--	38	2	7	--	73
	PG Flts	0	--	--	715	61	581	--	861
1,040,000	CLK Time	.58	--	6.91	6.0	.16	3.49	--	5.35
	CPU Time	.02	--	.55	2.75	.08	1.33	--	1.67
	Dir I/O	0	--	--	39	3	30	--	80
	PG FLTS	0	--	--	729	61	617	--	866

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #9

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE		SEED		RIM
		UFI	ORAAAA	GARDEN	HARVEST	TERM. INTER
52,000	Clock Time (SEC)	6.29(6.50)*	5.0 (6.6)*	2.18	2.44	--
	CPU Time (SEC)	.51(.49)	2.27(2.13)	1.00	1.26	--
	Direct I/O	--	30 (28)	5	5	--
	Page Faults	--	703 (660)	445	464	--
99,000	CLK Time	6.83*	--	2.06	2.28	3.55
	CPU Time	.51	--	.91	1.24	1.64
	DIR I/O	--	--	5	5	78
	PG FLTS	--	--	442	430	838
189,000	CLK Time	6.91*	6.91*	2.10	2.23	3.37
	CPU Time	.56	2.08	92	1.17	1.50
	Dir I/O	--	22	5	5	78
	PG. Flts	--	634	403	426	846
439,000	CLK Time	6.24	5.0	2.07	5.08	5.42
	CPU Time	.50	2.28	.93	1.64	2.37
	Dir I/O	--	30	5	5	66
	PG Flts	--	704	403	539	834
1,040,000	CLK Time	6.29	5.0	2.39	2.41	4.51
	CPU Time	.49	2.51	.99	1.26	1.49
	Dir I/O	--	30	5	5	73
	PG FLTS	--	725	445	475	846

* ORACLE Version 2.2 Used To Produce Results

BENCHMARKING OPERATION RESULTS #9

DATA BASE SIZE (Records or Rows)	MEASUREMENT	ORACLE		SEED		RIM
		UFI	ORAAAA	GARDEN	HARVEST	TERM. INTER
52,000	Clock Time (SEC)	6.29(6.50)*	5.0 (6.6)*	2.18	2.44	--
	CPU Time (SEC)	.51(.49)	2.27(2.13)	1.00	1.26	--
	Direct I/O	--	30 (28)	5	5	--
	Page Faults	--	703 (660)	445	464	--
99,000	CLK Time	6.83*	--	2.06	2.28	3.55
	CPU Time	.51	--	.91	1.24	1.64
	DIR I/O	--	--	5	5	78
	PG FLTS	--	--	442	430	838
189,000	CLK Time	6.91*	6.91*	2.10	2.23	3.37
	CPU Time	.56	2.08	92	1.17	1.50
	Dir I/O	--	22	5	5	78
	PG. Flts	--	634	403	426	846
439,000	CLK Time	6.24	5.0	2.07	5.08	5.42
	CPU Time	.50	2.28	.93	1.64	2.37
	Dir I/O	--	30	5	5	66
	PG Flts	--	704	403	539	834
1,040,000	CLK Time	6.29	5.0	2.39	2.41	4.51
	CPU Time	.49	2.51	.99	1.26	1.49
	Dir I/O	--	30	5	5	73
	PG FLTS	--	725	445	475	846

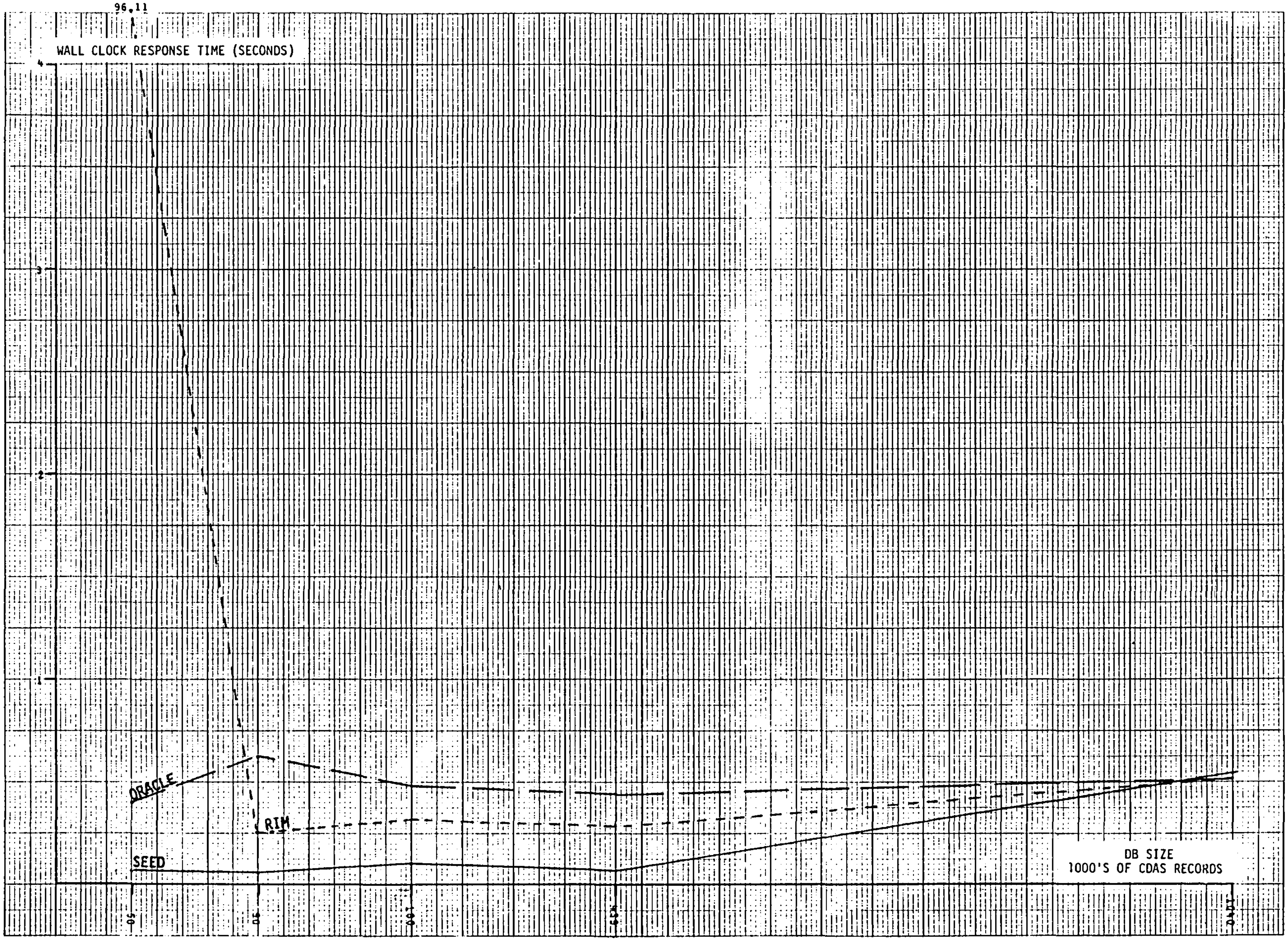
* ORACLE Version 2.2 Used To Produce Results

APPENDIX I

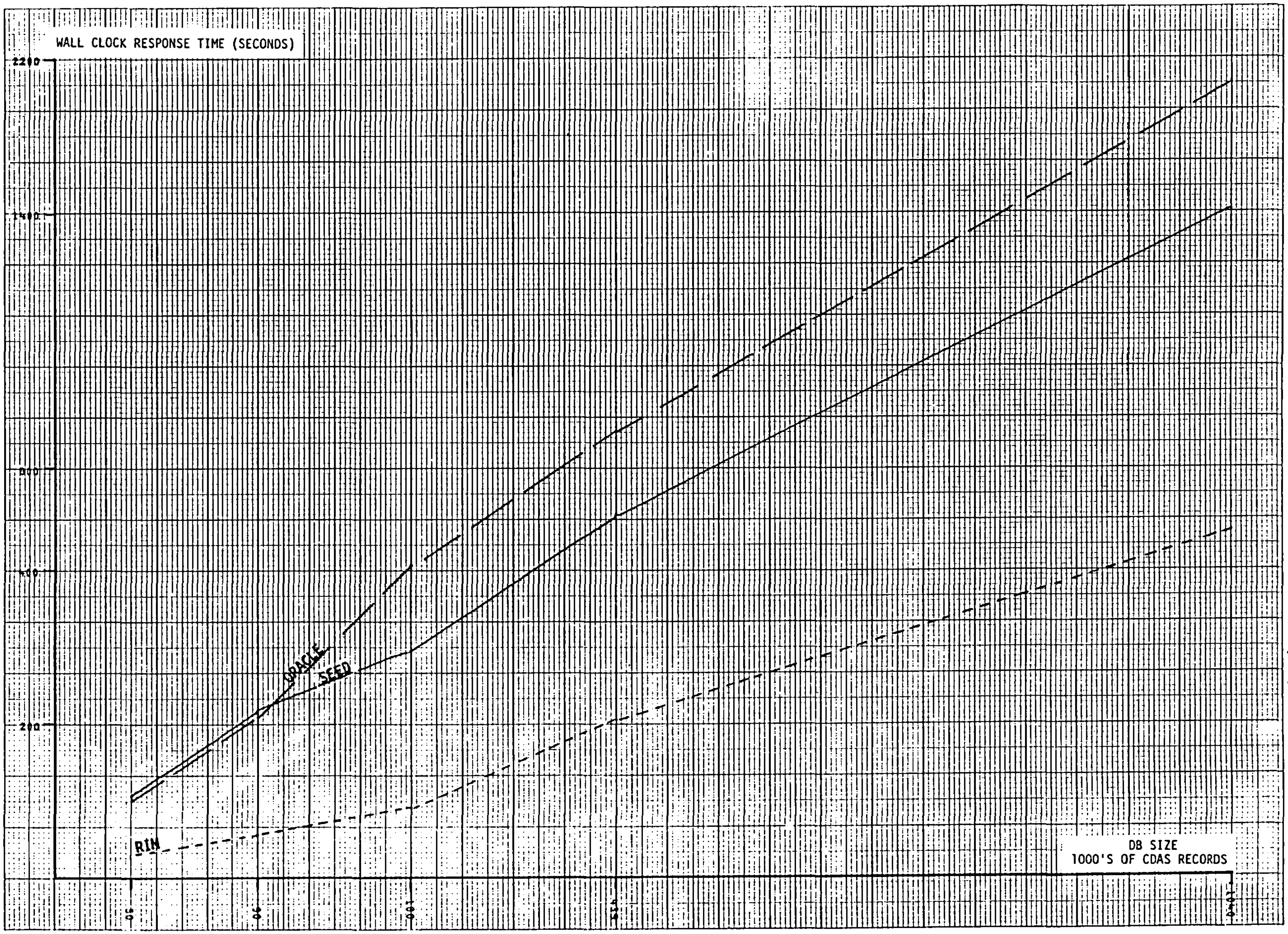
PART C

GRAPHS OF HLI BENCHMARK RESPONSE TIMES

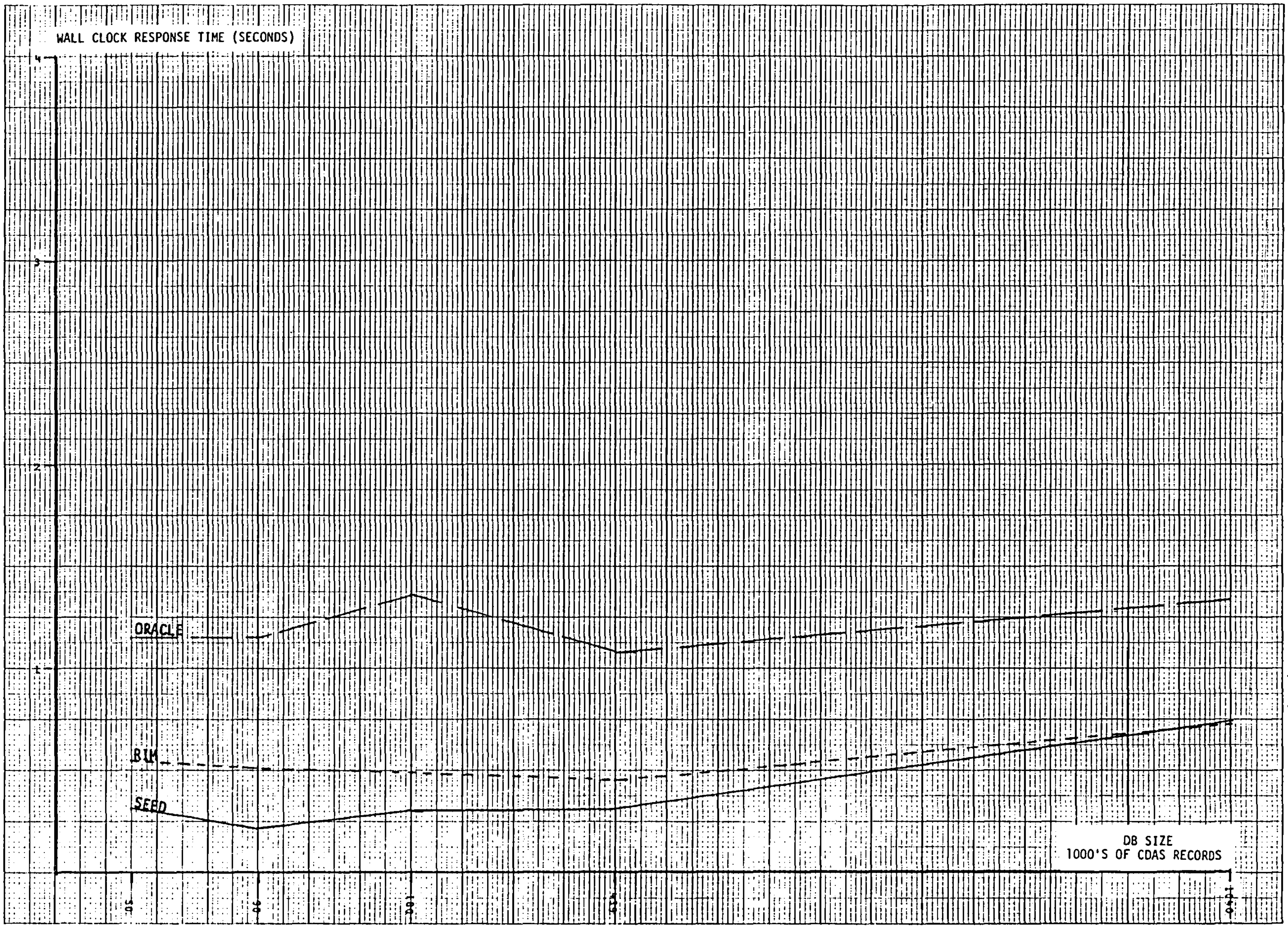
INDEX FIELD ACCESS OF PROFILE DATA (#1)



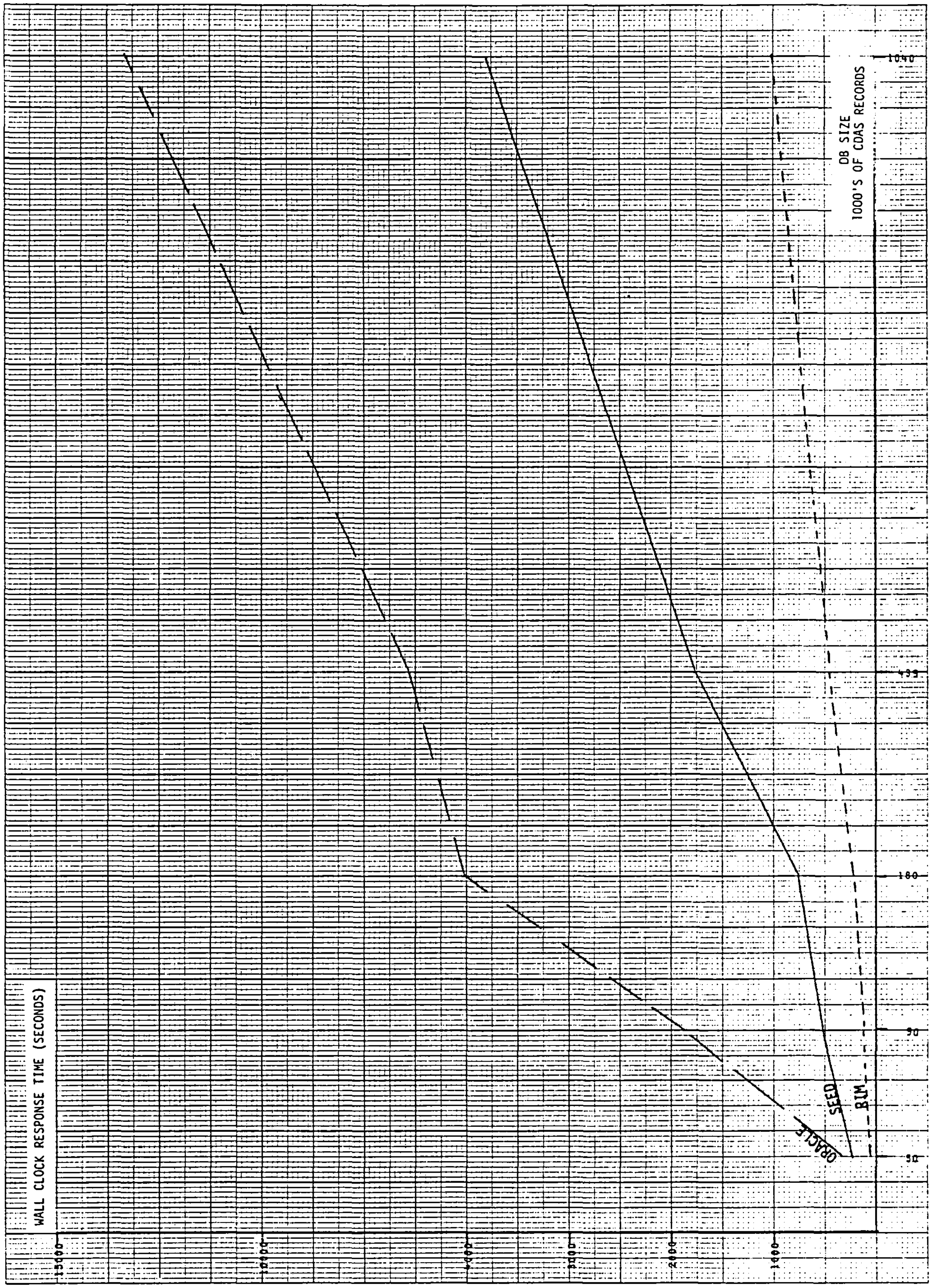
NON-INDEX FIELD ACCESS OF PROFILE DATA (#2)



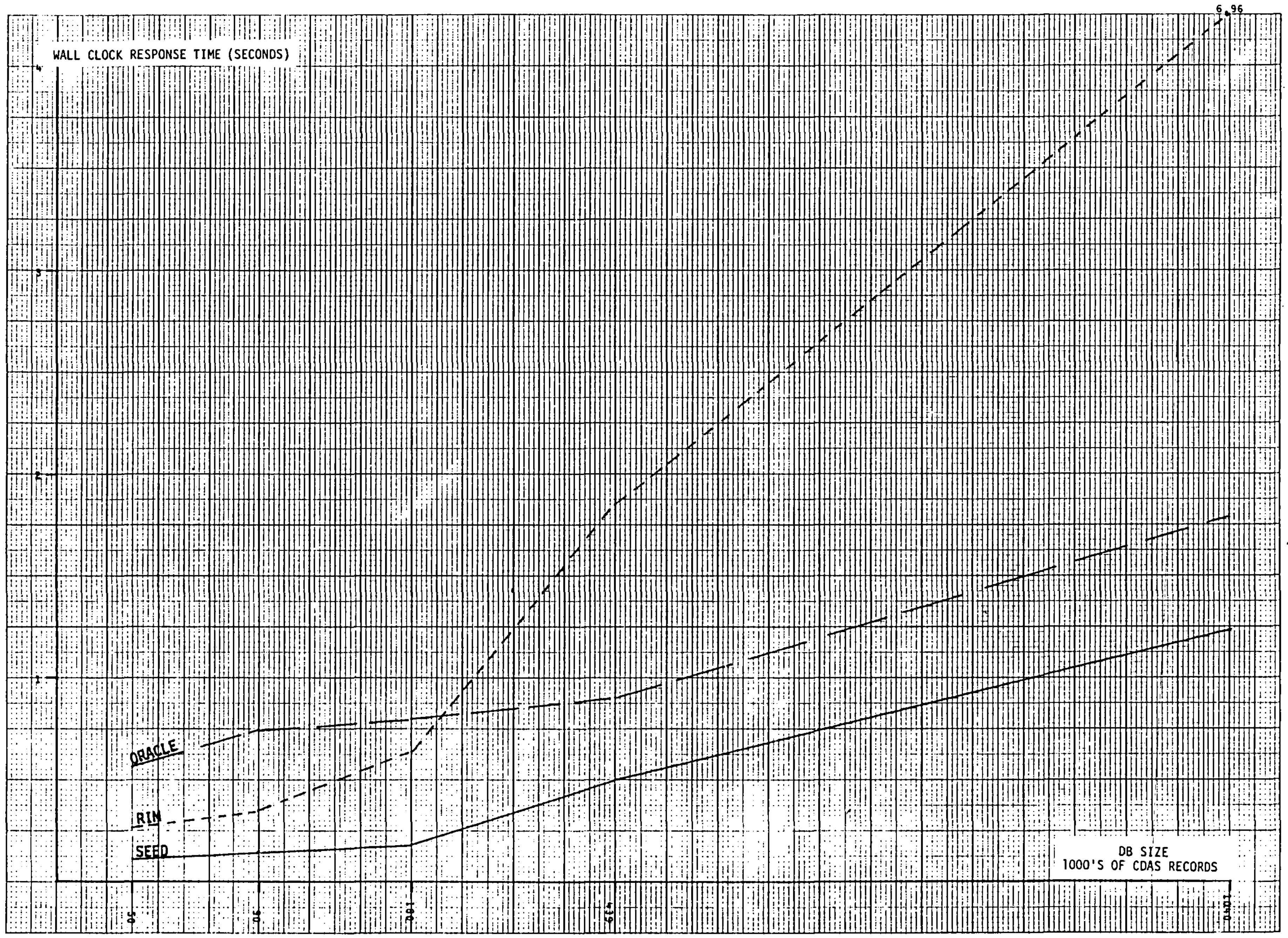
INDEX FIELD ACCESS OF ENTRY LEVEL DATA (#3)



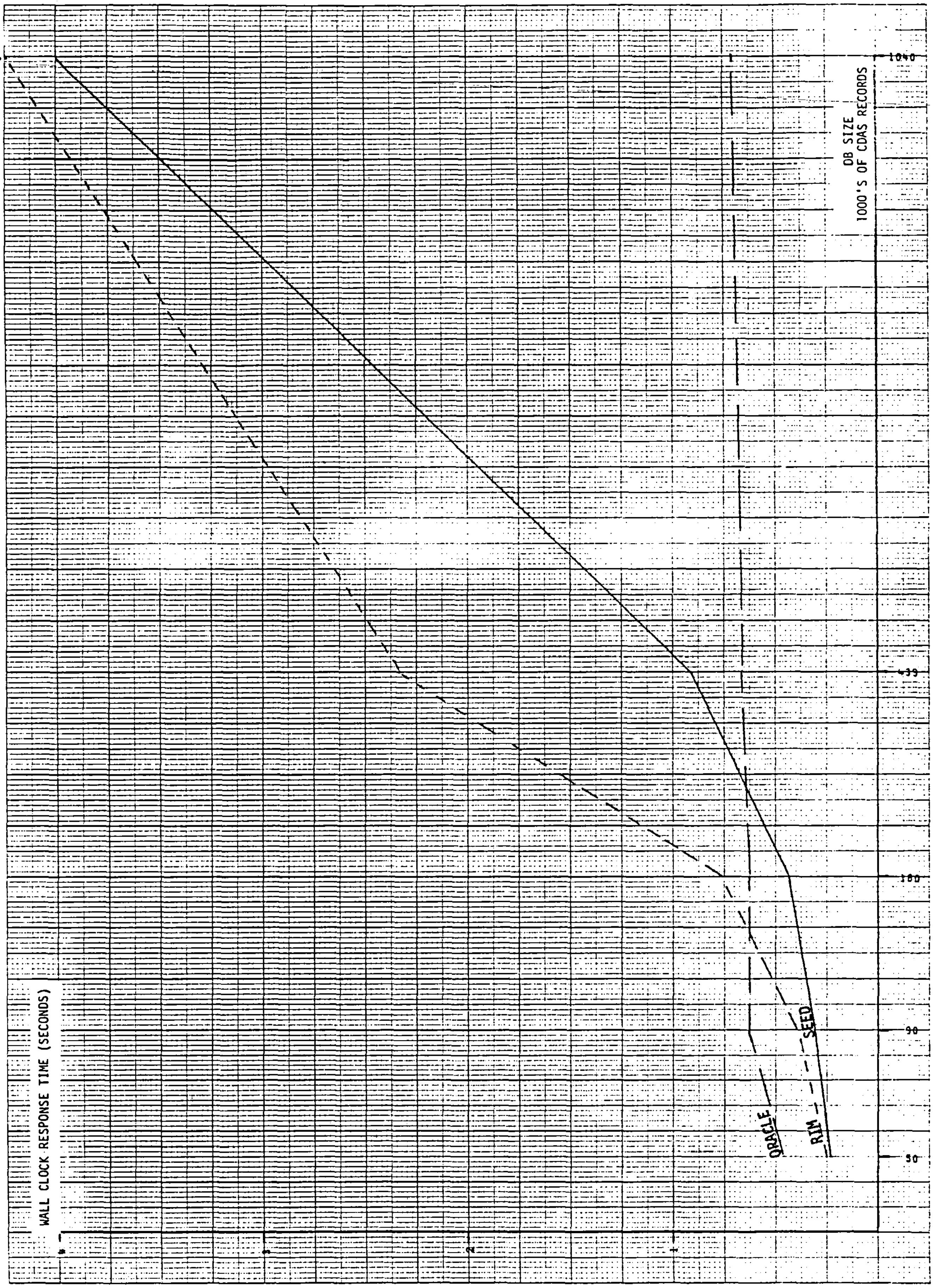
NON-INDEX FIELD ACCESS OF ENTRY LEVEL DATA (#4)



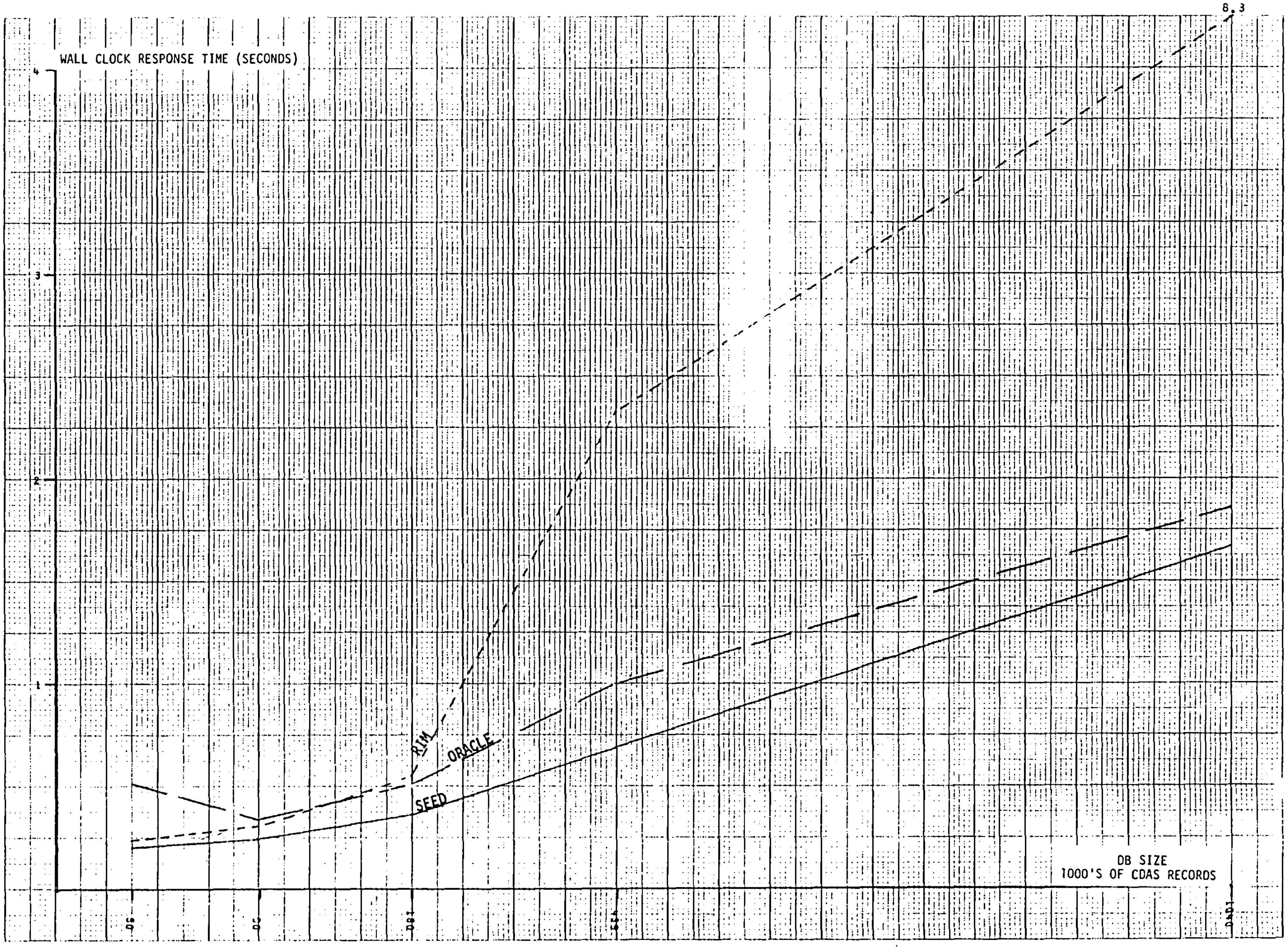
COMPLEX SEARCH CRITERIA (CONTROL #5A)



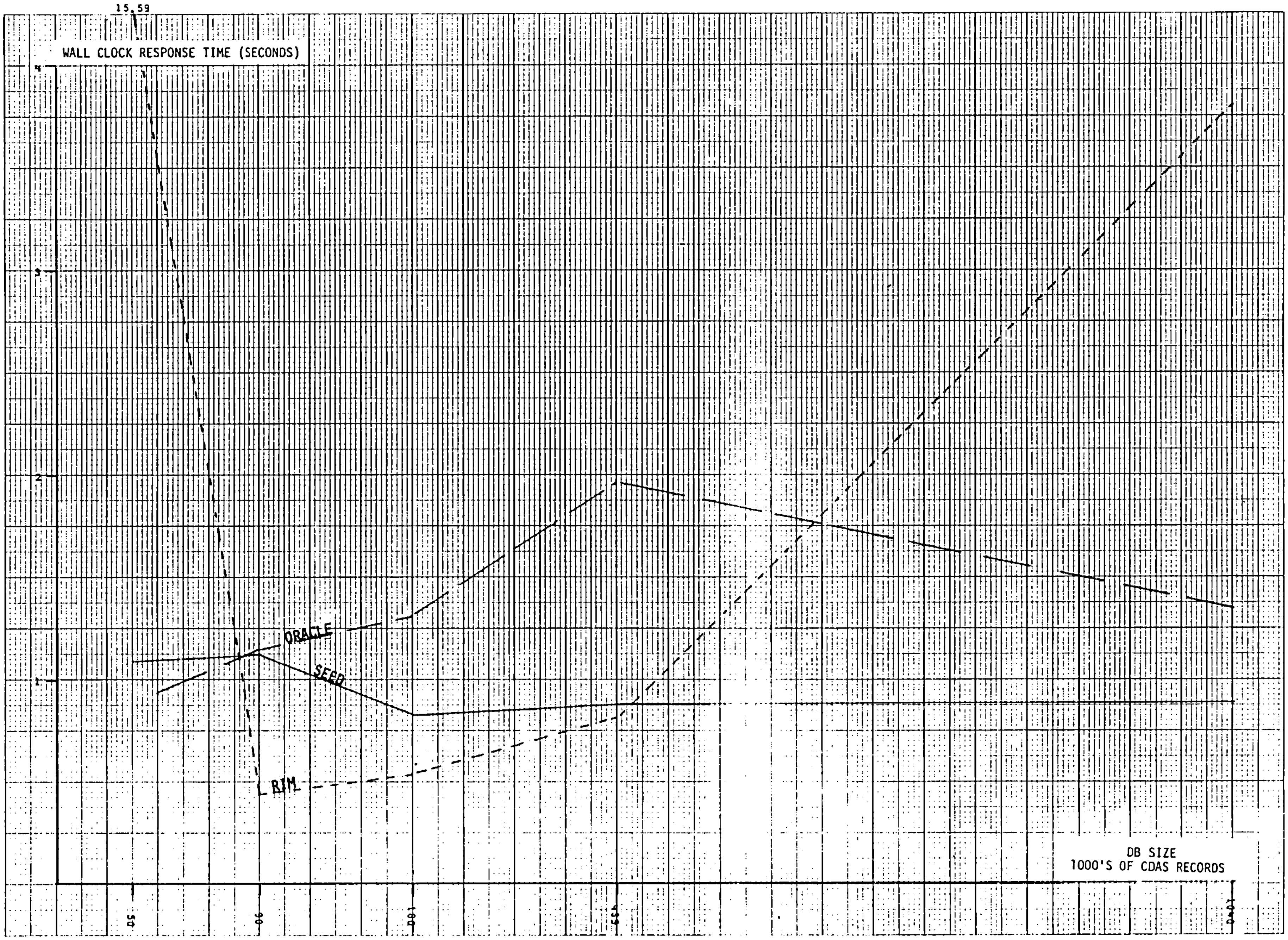
COMPLEX SEARCH CRITERIA (COMPOUND INDEX FIELD #5B)



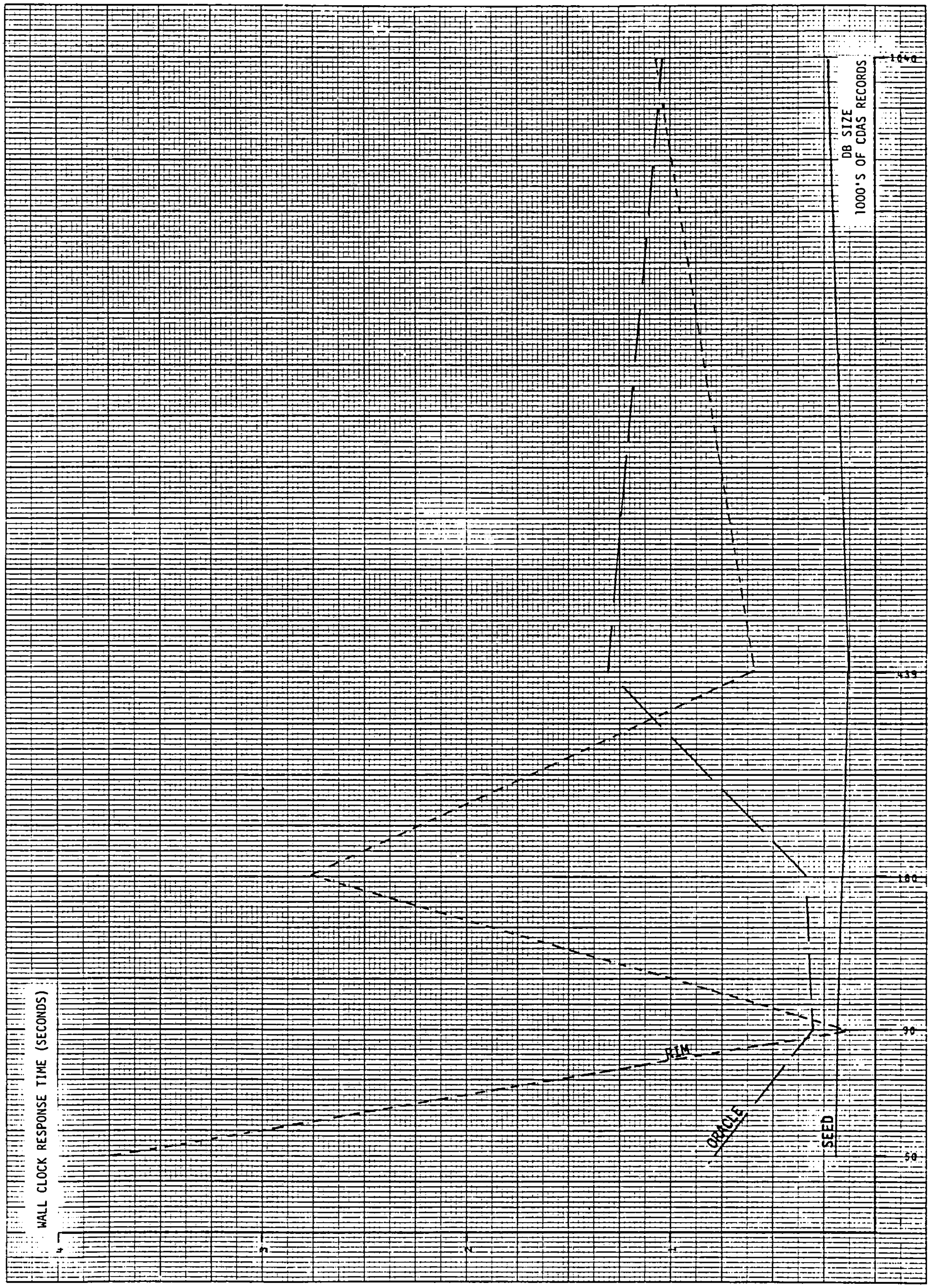
COMPLEX SEARCH CRITERIA (INDEXED & NON-INDEXED FIELDS #5C)



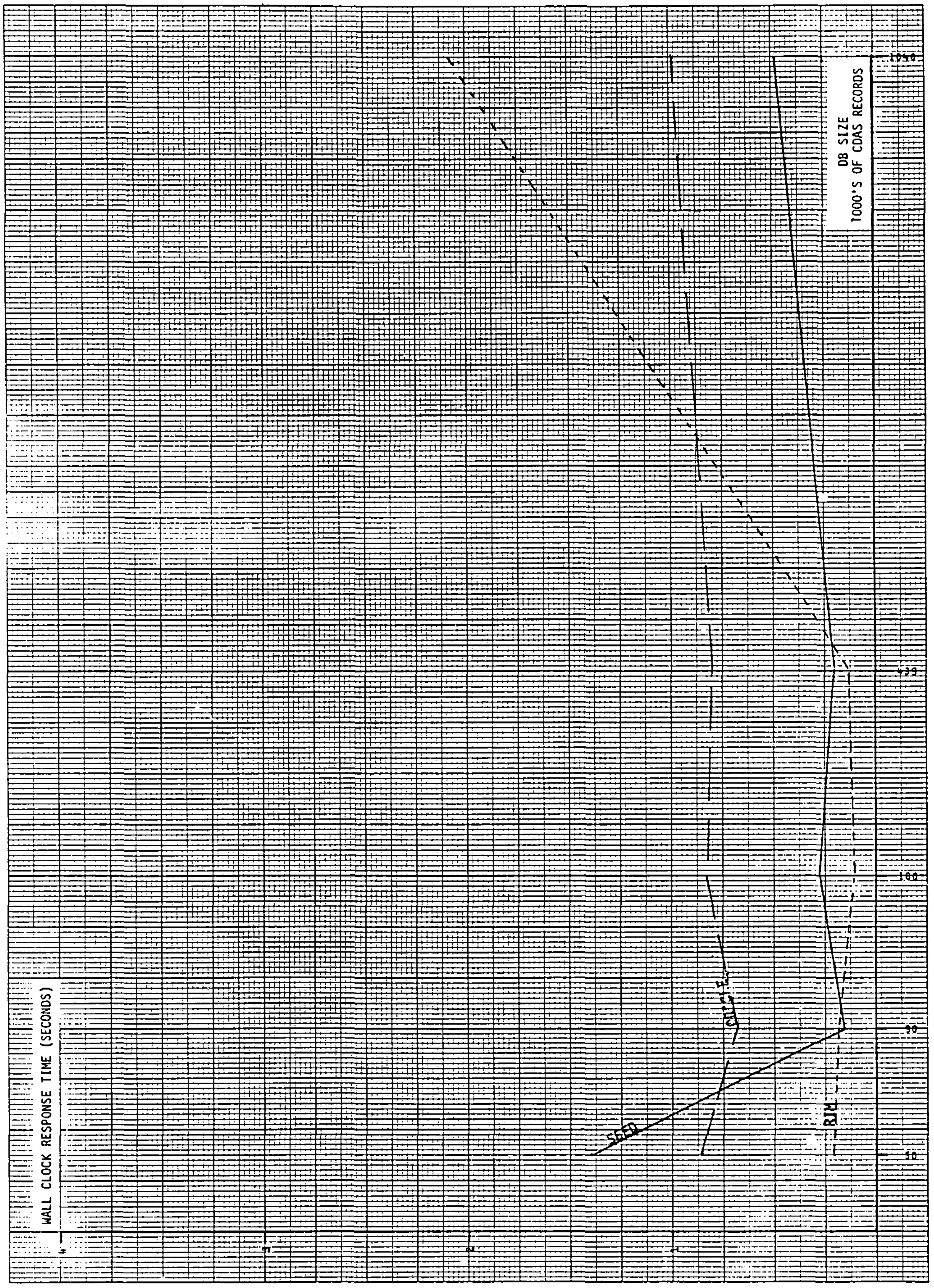
INCREMENTAL ADDITION OF A PROFILE RECORD (#7A)



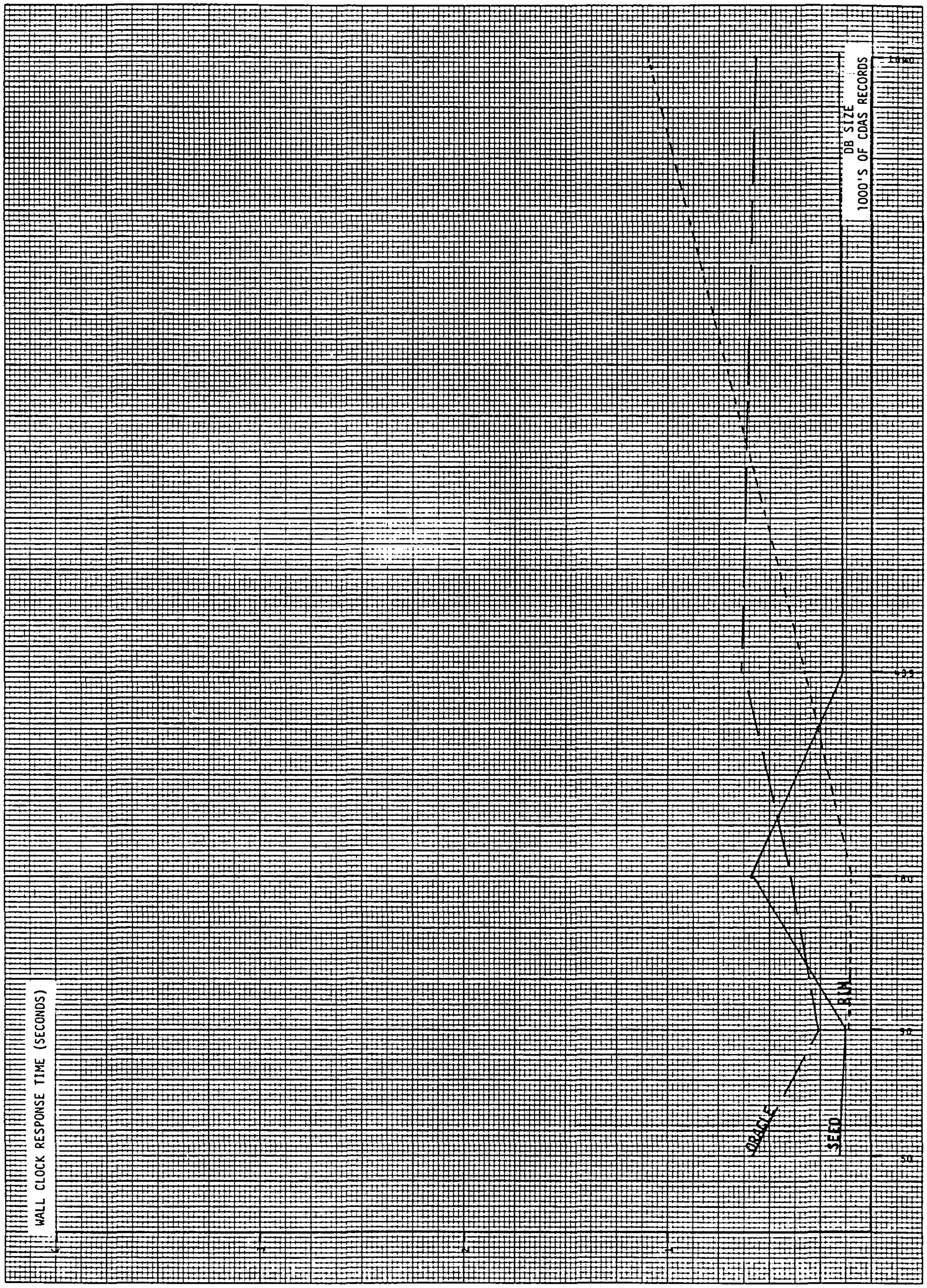
DELETION OF A PROFILE RECORD (#7B)



INCREMENTAL ADDITION OF AN ENTRY LEVEL RECORD (#8A)



DELETION OF AN ENTRY LEVEL RECORD (#8B)



APPENDIX II
DATA BASE SPECIFICATIONS

TABLE	COLUMN	DATE TYPE	LENGTH	IMAGE	NONULL
ENTRY	PROFILE_CNT#	NUMBER	22	NON-UNIQUE	YES
	PRESSURE_TYPE	NUMBER	22		YES
	PRESSURE_LVL	NUMBER	22		YES
	TEMP	NUMBER	22		YES
	QC_FLAG	NUMBER	22		YES
PRESSURE_TYPE_LEGEND	CODE	NUMBER	22	NON-UNIQUE	YES
	DESCRIPTION	CHAR	50		NO
PROFILE	PROFILE_CNT#	NUMBER	22	UNIQUE	YES
	TAPE_ID	CHAR	6		YES
	P_TIME	CHAR	10	NON-UNIQUE	YES
	LAT	NUMBER	22	NON-UNIQUE	YES
	LONG	NUMBER	22	NON-UNIQUE	YES
QUALITY_FLAG_LEGEND	CHAR	NUMBER	22	NON-UNIQUE	YES
	CODE	NUMBER	22		NO
	DESCRIPTION	CHAR	50		NO
TAPE	TAPE_ID	CHAR	6	UNIQUE	YES
	SYN_STIME	CHAR	8		YES
	SYN_ETIME	CHAR	8		YES
	GEN_DATE	CHAR	15		YES

APPENDIX II
PART A
ORACLE TABLE AND ROW
SPECIFICATIONS
FOR BENCHMARK TESTING

```
SCHEMA NAME SATDB MAXIMUM OF 36 RECORDS PER PAGE.
AREA NAME SATDATA AREA SIZE IS 64007 PAGES.
                  PAGE SIZE IS 512 WORDS.
AREA NAME SATHEAD AREA SIZE IS 2161 PAGES.
                  PAGE SIZE IS 256 WORDS.

RECORD NAME R1_SYN_STIME
       LOCATION MODE CALC USING SYN_STIME WITHIN SATHEAD.
            SYN_STIME       TYPE       CHARACTER 8.

RECORD NAME R2_SYN_ESTIME
       LOCATION MODE CALC USING SYN_ETIME WITHIN SATHEAD.
            SYN-ETIME       TYPE       CHARACTER 8.

RECORD NAME R3_TAPEID
       LOCATION MODE CALC USING TAPEID WITHIN SATHEAD.
            TAPEID          TYPE       CHARACTER 6.
            GENDATE         TYPE       CHARACTER 6.

RECORD NAME R4_DATATYPE
       LOCATION MODE CALC USING DATATYPE WITHIN SATHEAD.
            DATATYPE        TYPE       CHARACTER 30.

RECORD NAME R7_LAT
       LOCATION MODE CALC USING LAT WITHIN SATHEAD.
            LAT             TYPE       INTEGER*4.

RECORD NAME R8_LONG
       LOCATION MODE CALC USING LONG WITHIN SATHEAD.
            LONG            TYPE       INTEGER*4.

RECORD NAME R10_PROFILE
       LOCATION MODE CALC USING P_TIME WITHIN SATDATA.
            P_TIME          TYPE       CHARACTER 10.

RECORD NAME R11_ENTRY
       LOCATION MODE VIA S10_11 WITHIN SATDATA.
            PRESSURE_TYPE   TYPE       INTEGER*2.
            PRESSURE_LVL    TYPE       INTEGER*4.
            TMP             TYPE       INTEGER*2.
            QCFLAG          TYPE       INTEGER*2.
```

APPENDIX II
PART B
SEED SCHEMA
FOR BENCHMAR TESTING

(CONT'D)

```
RECORD NAME R100_QC_DESCR
       LOCATION MODE CALC USING QCFLAG CODE WITHIN SATHEAD.
            QCFLAG CODE     TYPE       INTEGER*2.
            QC_DESCR        TYPE       CHARACTER 60.

RECORD NAME R101_PRESSURE_TYPE_CODE
       LOCATION MODE CALC USING PRESSURE_TYPE_CODE WITHIN SATHEAD.
            PRESSURE_TYPE_CODE TYPE INTEGER*2.
            PRESSURE_DESCR TYPE CHARACTER 60.

SET NAME S1_3 MODE CHAIN ORDER NEXT
       OWNER R2_SYN_ESTIME
       MEMBER R3_TAPEID LINKED TO OWNER
       SET SELECTION CURRENT.

SET NAME S4_3 MODE CHAIN ORDER NEXT
       OWNER R4_DATATYPE
       MEMBER R3_TAPEID LINKED TO OWNER
       SET SELECTION CURRENT.

SET NAME S3_10 MODE CHAIN ORDER NEXT
       OWNER R3_TAPEID
       MEMBER R10_PROFILE LINKED TO OWNER
       SET SELECTION CURRENT.

SET NAME S7-10 MODE CHAIN ORDER NEXT
       OWNER R7-LAT
       MEMBER R10_PROFILE LINKED TO OWNER
       SET SELECTION CURRENT.

SET NAME S10_11 MODE CHAIN ORDER NEXT
       OWNER R10_PROFILE
       MEMBER R11_ENTRY
       SET SELECTION CURRENT.
```

APPENDIX III

COMMENTS OF STUDY BY DBMS ORIGINATORS

The following comments are responses by the originators of the DBMS's under study concerning the material presented in this document. The authors of this document do not confirm or refute the claims of the originators of the DBMS products on the pages that follow. Futhermore the authors wish to point out that any new statistics provided in this appendix by product originators have not been substatiated by the study group.

RSI

Relational Software Incorporated 3000 Sand Hill Road, Menlo Park, CA 94025 Telex-171437 415/854-7350

October 28 1981

NASA Goddard Space Flight Center
ATTN: Ms. Beth Martin
Code 931
Bldg 28 Room W-246
Greenbelt, MD 20771

Reference: NASA Study
"Database Management System Analysis and
Performance Testing with Respect to NASA Requirements"
Preliminary draft dated August 5, 1981.

Gentlemen,
Thank you for providing us the opportunity to review and comment on the referenced report, which presents the results of tests conducted with the ORACLE relational database management system as well as RIM and SEED. We were pleased to work closely with NASA personnel during the test program that is described, and we find that in general, the report is both balanced and technically correct.

Rather than comment at length, we have chosen to confine our response to two key aspects of the quantitative analysis section of the report, and one aspect of the qualitative analysis section.

First, the series of tests reported exercised each of the systems only in single user mode. We do not know why the tests were conducted in this fashion, since multi-user operation is critically important to large-scale use of a database management system. However, this mode of testing may have been selected because only ORACLE allows multiple users to update the database concurrently; SEED and RIM are limited to a single user for update. The tests discussed here did not utilize ORACLE's multiuser capability. For example, ORACLE allows two separate programs to concurrently load data into the same table.

The second point we wish to emphasize is that the particular version of ORACLE tested by NASA (i.e., Version 2.3) runs on the VAX 11/780 in PDP-11 compatibility mode; both SEED and RIM run in VAX native mode. Version 3 of ORACLE, now in test at RSI runs entirely in VAX native mode and is substantially faster than ORACLE version 2.3. (Version 3 ORACLE is scheduled for release to beta test sites in December of 1981.)

RSI has recently completed performance testing on ORACLE version 3 on the VAX 11/780. The following chart describes the performance of ORACLE version 1, (released in June 1979), version 2 (released in January 1980), and version 3 (scheduled for general release in March of 1982).

continued

O R A C L E   P E R F O R M A N C E

	VERSION 1	VERSION 2	VERSION 3	VERSION 3 CLUSTERED
SELECT	10/sec	50/sec	100/sec	175/sec
UPDATE	4/sec	15/sec	25/sec	40/sec
INSERT	3/sec	8/sec	22/sec	50/sec
DELETE	2/sec	7/sec	20/sec	35/sec

In addition to running in native mode, ORACLE version 3 includes among its enhancements a new physical storage optimization called 'clustering.' Clustering allows data from separate tables to be stored in the same physical disk page or block. The use of clustered storage is especially effective in optimizing the performance of join, insert and delete operations.

It is interesting that even though ORACLE was running in compatibility mode and without clustered storage, the NASA test demonstrated that version 2.3 ORACLE was faster than SEED on large databases, while SEED was faster on small databases. This result is somewhat ironic, considering that "experts" have been claiming that relational DBMS are not suitable for applications with large databases. We would be very interested in the result of a repetition of the tests using ORACLE version 3. We believe that version 3 will substantially outperform both SEED and RIM on large and small databases.

The qualitative analysis section of the report outlines the advantages of ORACLE's relational approach in the areas of ease of use and data independence compared with SEED's network data model. Unfortunately, the qualitative analysis did not include any ORACLE functions that SEED cannot also perform. That is, the test evaluated functions provided by both ORACLE and SEED even though the report states that ORACLE can perform many functions that either require large amounts of user programming or are not available at all with SEED. We hope that the next phase of testing will not be limited to that subset of functions of ORACLE that can also be performed by SEED.

Thank you again for your cooperation in letting us review the document before release. We look forward to cooperating with NASA in future tests.

Yours truly,

Lawrence J. Ellison

Lawrence J. Ellison
President
Relational Software Inc.

LJE/cba

INTERNATIONAL DATA BASE SYSTEMS, INC.
2300 Walnut Street, Philadelphia, PA 19103 (215) 568-2424
Suite 217

November 3, 1981

Ms. Beth Martin
NASA/Goddard Space FLight Center
Bldg. 26
Greenbelt, MD 20771

Dear Beth:

Attached is IDBS' letter of comment on the preliminary draft benchmark study — it essentially provides information on software released since this summer that should have some bearing on SEED's usefulness for NASA's satellite data cataloging requirements. Please feel free either to print the letter as a seperate appendix or to imbed the information in the body of your report. If further clarification is needed or the information might be more useful in another format, please don't hesitate to give me a call at (312) 787-6916.

IDBS would like to go on record as approving the direct comparison of benchmark results for the three systems tested — especially in the form of single graphs showing performance curves for all three systems. In addition, the Study carefully illustrates methodologies for determining data base file size, shows pre-load predictions of file size for both documented systems, and confirmed the high degree of accuracy of the predictive method. It might be of interest to the readers of the study to directly compare the predicted sizes for the 2 million CDAS record data base. The disk space requirements were: ORACLE 172 megabytes; SEED 40 megabytes. When very large data bases are anticipated, a difference in disk storage requirements of greater than a factor of four might have impact on the decision as to the appropriate means of implementation.

Attached are copies of IDBS' newest descriptive information on the SEED System, the C.0.0 release notes, and the most current VAX operating guide. I look forward to both the publication of this study and the results of further benchmarks by your group.

With best personal regards,

Evan [illegible]

Evan A. Bauer

encls.
EAB:ds

INTERNATIONAL DATA BASE SYSTEMS, INC.
2300 Walnut Street, Philadelphia, PA 19103 (215) 568-2424
Suite 217

November 3, 1981

Ms. Beth Martin
NASA/Goddard Space Flight Center
Bldg. 26
Greenbelt, MD 20771

Dear Ms. Martin:

International Data Base Systems, Inc. would like to thank the Information Extraction Division, Goddard Space Flight Center and Business and Technological Systems, Inc. for the careful job done in the study "Data Base Management System Analysis and Performance Testing with Respect to Nasa Requirements". We also welcome the opportunity to comment on the preliminary draft of the study. In particular, we would like to point out the areas where the new C.0 Release of the SEED Data Base Management System should make SEED more appropriate to the requirements of the Packet Management System (PMS).

Load Rates

Pages 1-5 and 1-6 in the Study compare the load rates of the three Data Base Management Systems used in the Study. IDBS concurrs with the Study in that the substantial degradation in load performance experienced is characteristic of the "lumping" of data caused by a poor match between calc key profiles and the hashing algorithm used to position the data in the data base. The cyclic change in storage rates would seem to indicate that "multiple hits" were resulting in long overflow chains that would consistently decrease performance until one of the "PROFILE" records was hashed to a page that was substantially empty. It should also be noted that setting the maximum number of records per page in the SATDATA area at 38 compounded the effect of the innapropriate hashing algorithm by causing additional data overflow that resulted in additional I/Os and the resultant increase in wall-clock time. IDBS provided the B.11.2 and B.11.3 SEED user with the option of writing up to nine additional hashing algorithms for the efficient handling of data not suited to the action of the SEED default algorithm.

With SEED version B.11.9, and subsequent releases, IDBS is now distributing several hashing algorithms, one of which is designed to randomly distribute multi-word keys like the time-stamp used as the calc key in the "PROFILE" record. It is IDBS' expectation that if the tests were repeated using this new standard algoritnm that the SEED load-rate curve would decline much less sharply and show much less fluctuation.

We fear that the language in the second paragragh on page 4-2 may mislead some readers. Although it is clear from both the first paragraph and figure 4-1 that at one million records, SEED's performance is approximately 25% better than that of ORACLE, the second paragraph could mislead a reader into believing that ORACLE's performance in loading a database of 700,000 or more records was better. Perhaps it would be less misleading to state that in the 700,000 to 1,000,000 record range, ORACLE's marginal performance was better than SEED.

HARVEST

Enhancements have been made to the HARVEST Query Language since the B.11.3 version that was used for the Study. In response to the requirement to use an escape sequence to abort an incomplete incorrect query, IDBS has now added the ampersand (&) character to HARVEST as an abort character. The insertion of an ampersand character at the end of a HARVEST command string aborts the command and returns the user to the HARVEST "COMMAND" prompt level.

In addition, the query diagnostic capability of HARVEST has been substantially enhanced to assist the user in identifying syntactical or logical errors in his query and to correct them without requiring the use of any reference documentation.

HARVEST now has a sort capability equivalent to that in the BLOOM Report Writer that allows multiple ascending and descending sort keys to be specified in a single query using a simple addition to the DISPLAY clause (SORTED on (field-name) (ASCENDING/DESCENDING)).

New Products

Descriptions of VISTA and RAINBOW new addition to IDBS' SEED System, might be appropriate additions to section 3.7.2 (Complementary Software for SEED) of the Study.

RAINBOW is an interactive graphic display facility designed to be used by technical and non-technical users of SEED-managed data bases. It makes it easy for users to obtain pie charts, bar graphs and line graphs in multiple colors. RAINBOW provides a high degree of automation, but at the same time, sufficient flexibility for directing, modifying, and manipulating the presentation of information. RAINBOW acts as graphic designer and graphic artist in response to user queries. RAINBOW displays query output in a graphic format in black and white or color in accordance with the user's objectives.

RAINBOW is a modular addition to the HARVEST System. It provides the same degree of default capability in its automatic graphic displays, as HARVEST provides in its tabular displays. Use of the "DRAW" command invokes RAINBOW. Input data is then internally structured to relate to design attributes of shape, color, and scale.

VISTA is a screen-oriented applications development system which permits a user to create routines to accept, edit, retrieve, and process data. It is device-independent, and will run on virtually any video terminal that supports a minimal set of cursor movement functions. VISTA offers non-procedural screen definition, full SEED Data Manipulation Language Capabilities, the capability to perform arithmetic calculations, and "if" testing on data that is entered, generated, or retrieved. VISTA allows the user to take advantage of such terminal features as reverse video, direct cursor addressing, and graphical capabilities (where available) to produce "user-friendly" screen formats. Because VISTA is dictionary-driven, the user need not explicitly describe the syntax of data base items. VISTA is fully integrated with SEED´s journaling function, allowing screen controlled backout of erroneous transactions.

VISTA can be run as a stand-alone interpretive processor or as a set of subroutines callable from FORTRAN.

The C.0 version of SEED supports the use of "b-tree" managed pointer arrays for accessing data. The pointer arrays can be used instead of linked lists to implement sorted sets (changing performance characteristics but not affecting the users´ view of the data base); or can maintain seperate indices to any field in a record. A field can be indexed after the data base has been designed and loaded. HARVEST´s automatic navigation capabilities can make use of indices and pointer array maintained sets. The users´ "relational" or flat-file view of the SEED data base need not be changed.

IDBS would again like to thank all of the organizations involved in this study and we look forward to both the publication of this study and the results of further benchmarks by NASA/IED.

Sincerely yours,

[illegible]

Rob Gerritsen
President

encls.
RG:ds

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia
23665

NASA

NOV 2 1981

Reply to Attn of: MS246(L-1155-REF)

TO: Goddard Space Flight Center
Attn: 931/Regina Sylto, Information Extraction Division

FROM: 246/IPAD Project Manager, IPO, SDD

SUBJECT: Performance of Data Base Management Systems

Thank you very much for a copy of the draft report "Data Base Management System Analysis and Performance Testing with Respect to NASA Requirements," forwarded with your letter of September 29, 1981.

The report contains a lot of data and its careful review will require more time than your deadline of October 12 permits. Some initial comments relative to RIM are as follows:

1. A RIM-5 is already operational and a copy is available from COSMIC. Selected other enhancements are also being added consistent with its basic capability and scope.

2. A multi-user version of RIM is currently operational; however, the documentation has not yet caught up with the process.

3. RIM is being developed as part of a CAD R&D effort to develop technology for management of engineering information, and we would recommend these words be included in the front of the report. Its development and evaluation is an ongoing part of the IPAD project. RIM currently runs on DEC, PRIME, CDC, IBM, UNIVAC, and CRAY computers. While not production software, several organizations now have production versions in operation.

Thank you for your constructive comments on RIM. We will transmit a copy of this draft report to the IPAD Program Office at The Boeing Company to guide future RIM enhancements and to obtain any technical comments.

[illegible]
Robert E. Fulton

RELATIONAL INFORMATION MANAGEMENT (RIM)

- INITIALLY EXPERIMENTAL SOFTWARE
- IN-CORE CYBER VERSION 1978
- PAGING CYBER VERSION 1980
- VAX VERSION 1980
- MULTIHOST VERSION (CDC, DEC, IBM, UNIVAC,PRIME)
  - AVAILABLE OCT 1981
    - MAJOR ENHANCEMENTS
      - SCIENTIFIC DATA ATTRIBUTES
      - VARIABLE LENGTH ATTRIBUTES
      - FORTRAN INTERFACE
      - RIM TO RIM INTERFACE

PRODUCT AVAILABILITY

- PLANNING, REQUIREMENTS, PD DOCUMENTATION
- INTEGRATION PROTOTYPE
- RIM (MULTIHOST VERSION)
- MONTHLY TECHNICAL PROGRESS NARRATIVE
- REQUEST FROM

  D. E. TAYLOR
  IPAD PROGRAM SUPPORT MANAGER
  MS 73-03
  P.O. BOX 24346
  SEATTLE, WA 98123

  TELEPHONE (206) 237-2389

RIM-5 DEVELOPMENT STATUS

● CYBER NOS VERSION UNDER PROGRAM CONFIGURATION CONTROL	10/81
• CHECKED CODE	
• DOCUMENTATION	
• INSTALLATION TESTS	
• AVAILABLE FOR DISTRIBUTION	10/81
● CYBER NOS/BE VERSION	
• CONVERSION BY GENERAL DYNAMICS/CONVAIR	8/81
• DOCUMENTATION	10/81
• TRIAL INSTALLATION AT GENERAL DYNAMICS	10/81
• AVAILABLE FOR DISTRIBUTION	11/81
● VAX VMS VERSION	
• CONFIGURATION CONTROL	10/81
• AVAILABLE FOR DISTRIBUTION	10/81

RIM-5 DEVELOPMENT STATUS (CONTINUED)

- UNIVAC EXEC VERSION

• CONVERSION BY LOCKHEED, GEORGIA	8/81
• DOCUMENTATION	10/81
• TRIAL INSTALLATION AT LOCKHEED GEORGIA	11/81
• AVAILABLE FOR DISTRIBUTION	11/81

- PRIME PRIMOS VERSION

• CONVERSION BY NASA	9/81
• DOCUMENTATION	11/81
• TRIAL INSTALLATION	11/81
• AVAILABLE FOR DISTRIBUTION	11/81

- IBM VERSION

• FORTRAN 66 VERSION AT BOEING (VM/CMS)	9/81
• FORTRAN 77 VERSION CONVERSION AT GENERAL DYNAMICS/CONVAIR AND NORTHROP	IN PROGRESS
• DISTRIBUTION	UNDETERMINED

RIM POTENTIAL FUTURE ENHANCEMENTS

- FUNCTIONALITY
  - REPORT WRITER
  - PLOT INTERFACE
  - EDIT OF ATTRIBUTE VALUES
  - ATTRIBUTE UNITS, DESCRIPTION, ALIAS
  - ARITHMETIC CAPABILITY IN COMPUTE COMMAND
- PERFORMANCE
  - KEY PROCESSING
  - I/O ON SPECIFIC HARDWARE
  - IMPROVED SORT
- MULTI-USER, TRANSACTION PROCESSING
  - DESIGN FOR MULTI-USER
  - IMPLEMENTATION OF MULTI-USER ON SELECTED HARDWARE

APPENDIX IV

The following is a reprint of Chapter Three of "NEEDS Data Base Management System Functional Requirements" dated March 20, 1980, written by J. Patrick Gary, Karen W. Posey, and Ronald W. Durachka for the Information Extraction Division at the Goddard Space Flight Center in Greenbelt, Maryland. This chapter summarizes the original requirements of the Integrated Data Base Management System (IDBMS) and as such provided guidance to the study. For additional information the reader is referred to the document itself.

3. REQUIREMENTS

3.1 Functions

3.1.1 Data Reception and Storage

1. Manage a dynamic local archive consisting of climate-related and other Applications and Space Science data sets. These will consist of NASA and related scientific data including digital image data, satellite measurements, correlative ground truth data, results of application data analysis programs, extracted parameters, application modeling results, and information describing data characteristics and data sets (see Section 3.3 and 3.4). The local archive will contain mostly derived geophysical parameter data and sensor radiance data from NASA missions; however, non-NASA data and non-satellite data must also be accommodated.

Provide for entry of data into the local archive via magnetic tape, disk pack, or telecommunications through a computer network (see Section 4). Data will be received in various mission-dependent sequential file formats. The data will be physically stored in the archive on magnetic tape, disk packs, or other direct access storage devices.

2. Manage a local archive of packetized data stored in the MSFC Archival Mass Memory (AMM) (see Reference 16). Packetized data will be entered by the MSFC DBMS into the AMM following its receipt via electronic transmission using the CCITT X.25 protocol (see Reference 16). Although this tramission and storage function will be handled external to the IDBMS, it must catalog and manage this archive as described below. Packetized data may also be entered into the AMM under the control of the IDBMS (see Section 3.1.1(3) and 3.1.3(4).

3. Accept, store, and manage data sets created by application programs running in the IDBMS environment or entered from local and remote interactive terminals. The data will be physically stored in the local

archive on magnetic tape, disk packs, the AMM, or other direct access storage devices.

4. Provide for the addition, deletion, and logical and physical replacement of data sets in the local archive.

5. Provide an interface for data producer specified software modules to perform data quality control checks. These data type dependent quality checks will be performed on data sets entered into the archive and/or created by the system and will include such tests as format checks, tape quality checks, and limit checks (i.e.,out-of-range values).

3.1.2 Catalogs

1. Provide for the flexible construction and maintenance of a catalog* of available climate-related data (see also 3.1.2(3)). The catalog will describe all climate-related data in the local archive, as well as certain climate-related data which is archived elsewhere and is not directly accessible to the system. This includes data sets created by the IDBMS as well as those created outside the IDBMS and transferred to the archive.

Data will be described in the catalog at various levels of aggregation. At the highest level of aggregation, a catalog entry could represent the collection of all data available for a specific measurement or parameter from a single source instrument and with a single physical storage format. For instance, all radiance data from the NOAA Scanning

*As used herein, "catalog" is a collective term referring to the total collection of information describing the characteristics and locations of available data. The architecture to be used in logically and physically partitioning this information is a system design consideration to be addressed as part of the system design phase.

Radiometer (SR) instrument, or all sea surface temperatures from the Nimbus-7 Scanning Multichannel Microwave Radiometer (SMMR) instrument could be considered a data set at this level. For data sets at this level of aggregation, the catalog will maintain summary information on all available data sets. More detailed descriptions and information about each data set will also be maintained.

Sample outlines for typical summary and detailed descriptions to be maintained in the catalog are given in Appendix A. A sample detailed data set description using the outline shown in the second figure of Appendix A is presented in Appendix B.

At the lowest level of aggregation, a catalog entry could represent a specific physical storage entity, such as a tape volume, a tape or disk file, a data packet, or a specific logical entity such as a gridded array (map) within a file. For each data set at this level, the catalog must maintain key descriptors such as name/ID, source, parameter(s) represented, time period of coverage, archival location, etc. A sampl listing of this type of catalog information is shown in th third figure of Appendix A.

These figures are intended as examples only. The IDBMS must provide the capability to dynamically define the contents and organization of these catalogs and the levels of data aggregation described by the catalog entries.

2. Provide for the automatic construction and maintenance of catalogs suitable for managing the storage and retrieval of packetized data. The catalog entries (packet descriptions) shall contain information extracted from the primary and secondary headers of packets. For data received by the MSFC DBMS (see Reference 16), the MSFC DBMS will make these packet headers available to the IDBMS as the packets are stored in the AMM. The IDBMS must examine these headers and automatically update catalog and cross-reference information. For packetized data sets created by the IDBMS (see Section 3.1.3(4)), the catalog entries will be constructed by the IDBMS from the packet header information provided with the data sets.

3. Provide for the flexible construction and maintenance of other Applications and Space Science data catalogs permitting different descriptions for other data or for new data types to be entered into the system.

4. Provide a capability for users to create their own catalogs describing locally archived data, data archived elsewhere, and user (system) created data sets.

5. Provide for the flexible construction and maintenance of "bibliographic catalogs" (vs. data catalogs) which describe supporting documents and applications software related to the archived data. These catalogs will contain information such as title/name, author, abstract or overview description, and physical location or reference number.

Provide the ability to cross-reference such documents and software to data sets maintained in the archive.

6. The catalog update capability shall include, where appropriate, automatic construction of catalog entries from information extracted from the data set. It shall also allow users to directly specify catalog entries or fields within the entries where appropriate.

7. Provide the capability to distinguish "master" and "secondary" data sets. As used herein, "master" data set refers to the primary version of a given data set (e.g., the "best" set of Nimbus-7 SMMR sea surface temperature available in the archive). "Secondary" data sets refer to other versions of data represented in the master sets (e.g., tapes created by copying portions of one or more master tapes, data sets derived via different versions of the retrieval algorithms, etc.).

8. Provide the capability to extract catalog-type descriptive information from a data set and to display or print this information without actually entering it into the catalog.

9. Provide catalog access, search, and display capabilities to enable users to determine what data is available and what data they want retrieved from the archive.

10. Provide the capability to limit catalog access/search to user-specified criteria related to the catalog entries, such as limited area coverage, specific parameters, time periods, or data sources. This includes efficient text-searching capabilities, where necessary, to locate appropriate catalog entries where fields (search values) are text data.

11. The catalog data base structures and the algorithms for searching/updating them must accommodate a flexible, dynamically definable set of catalog search criteria. They must be designed for maximum search/update efficiency, and allow for dynamic creation and removal of appropriate internal structures. These internal structures must be updated automati- cally by the system as the catalogs are modified.

12. The internal structures of the catalogs must be expandable and deletable without necessitating a reorganization or reloading of the catalogs. Expansion must allow for the addition and deletion of catalog fields, alteration of field size, the addition and deletion of search criteria/keys, additional values for the search criteria/keys, and additional catalog entries containing these values. The system must dynamically reuse available space created by deleting catalog information or internal system data. It must also be possible to add new data base catalogs without necessitating a reorganization of existing catalogs.

13. Provide a capability whereby a user can request a count of the number of data sets which satisfy a complex search condition prior to retrieving catalog information for such data sets.

14. Provide for variable length fields within the catalog entries, i.e., provide for the storage of variable length values for each item in a catalog description such that only the amount of space actually needed to

accommodate the information present in a catalog entry is used for its physical storage.

15. Provide capabilities which handle multi-valued fields within the catalog entries, i.e., for fields having two or more distinct values within a single entry. (An example of this requirement is Nimbus-7 SMMR digital maps, each of which contains multiple parameters.)

16. Provide capabilities which handle multi-valued groups of fields (repeating groups) within the catalog entries, i.e., for groups of logically related fields which occur more than once within an entry. (An example of this requirement is Nimbus-7 SMMR digital maps, where each parameter in a map has its own associated spatial resolution.)

17. Provide for null fields within the catalog entries, i.e., for catalog entries having no assigned value for a given field.

3.1.3 Data Access and Manipulation

1. Provide for user selection of, and access to, a subset of data in the local archive, distinguished by a flexible, dynamically definable set of characteristics of the data, including source, parameters, coverage (time, space) and level of reduction.

2. Provide flexible data manipulation and reorganization capabilities for data in the local archive. These capabilities should include sorting (time, location, parameter), interpolation and smoothing, averaging, histogramming, and generation of gridded data sets (for a limited set of projections and space/time grid intervals) for user-selected data in the local archive.

3. Provide capabilities to update, access and invoke a library of processing routines for performing selected general purpose data processing function.

4. Provide a capability to format a non-packetized data set into a packet formatted data set. Information needed to construct the necessary primary and secondary packet headers shall be obtained through the interactive user terminal or application program interface by which the packet formatting capability was invoked or, if the data set is already cataloged, from descriptors in the data set's catalog entry.

3.1.4 User Interface

1. Provide an interactive terminal user interface and an English-like user language for requesting the data and catalog input, update, selection/search, retrieval, manipulation and output capabilities. The user language must be easy to learn and use and must provide clear explanatory messages (not codes) following the input of any incorrect syntactic construction.

2. The user language must utilize menu-guided prompts whenever possible to facilitate the user selection and specification of processing options.

3. The user language must include data description/dictionary capabilities for defining and describing the properties of all data and catalog information stored in the data base. This shall include capabilities to:

- Assign names and descriptions to data and catalog elements/fields or groups of logically related elements and to describe their size and type of representation (alpha, numeric, etc.).

- Declare whether an element/field is a key or a non-key item.

- Describe the logical structure of the catalog or data base, i.e., the constituent elements/fields and their logical relationship to one another.

Retrive and display all data description/dictionary information stored in the system.

4. The user language must provide the capability for the user to request and display descriptions of what capabilities are available for data and catalog input, update, search, retrieval and output.

5. The user language must permit the dynamic definition, creation, and deletion of individual data sets and catalog entries.

6. The system must support the submission of IDBMS commands from batch input. The syntactic format of batch submitted IDBMS commands must be identical to those submitted via interactive user terminals.

7. The system must enable the user to write a sequence of IDBMS commands and store them on disk under a referenceable name. The user language must allow the user to execute the sequence of commands by appropriate reference to the command sequence name.

8. Provide an application program interface to the data and catalog input, update, selection/search, retrieval, manipulation and output capabilities. This program interface is oriented to high level procedural language applications software, and shall be FORTRAN callable.

9. System status indicators must be provided to an application program each time the system is given a command. The indicator(s) must allow the application program to determine, at a minimum:

- that the command to the system was understood and executed without detectable error, or

- that the command to the system was not error-free and the nature of the error involved, e.g., syntax error, invalid record reference, etc.

10. An application program must not require modification or recompilation when a catalog or data field is added, deleted, or changed in format or classification of content, unless that field is referenced in the application program. The addition or deletion of indices or other internal search structures must not require modification or recompilation of application programs.

11. The system must not require that interactive, batch, and application program users be aware of physical locations or storage structures of catalogs, data indices, pointers, or other internal structures.

12. Provide capabilities for applications programs to perform the following basic input/output and file maintenance operations for tape, disk, and the AMM: a) open and close files, b) read/write next record, c) read specified record, and d) copy or move files from one device to another.

3.1.5 <u>System Outputs</u>

1. Provide options to display catalog access/search results at local and remote alphanumeric CRT terminals, to print this information, or to output it to disk or computer compatible tape in user-specified formats.

2. Provide data output as disk files, computer compatible magnetic tape files (including reformatting, if necessary, to accommodate a limited set of magnetic tape characteristics), alphanumeric CRT and graphics terminal displays, printed listings and graphic plots, and image (raster) terminal display.

3. Provide a capability by which IDBMS created data products can be stored temporarily or permanently as master or secondary data sets managed by the system. (See 3.1.2.(7)).

3.1.6 System Operation, Control and Accounting

1. The IDBMS must operate in a multi-user, multi-thread mode wherein several users are processed concurrently. For example, when the request that the system is currently processing requires I/O, the system requests the I/O and then begins processing another request while waiting for the first I/O operation to be completed. The system must not require that one request be serviced to completion before the next one is begun. This shall include synchronous control to permit concurrent access to data catalogs and to archived data sets by multiple interactive terminal users, batch users, and application programs. The system shall permit up to 32 concurrently active interactive terminal users, batch users and attached application programs.

2. Provide user transparent lockout features at the lowest practical level to support multi-user access to data catalogs and to archived data sets and to prevent concurrent access/update problems while minimizing the degree to which a particular access/update operation "ties up" the data base.

3. Maintain system logs and provide accounting reports of data entered into and deleted from the local archive as well as all data processing activities performed by the system, i.e., data selected, accessed, manipulated and output. These logs will include descriptions of the activity performed, when it was performed, and who requested it.

4. Provide mechanisms (a) for supplying users with information (including costs) for data ordering, (b) for storing and updating accounting data on the ordering processes (who, what, when, costs), and (c) for producing usage reports from this data.

5. Provide a capability for IDBMS users to invoke VAX/VMS system utility routines (e.g., file dumps, file copy routines, etc.).

6. Provide capabilities which will permit changing the physical location (storage device or location within a storage unit) and reorganization of catalogs and data sets to minimize seek time, to consolidate free space, and to optimize the use of system resources.

7. Provide capabilities whereby the Data Base Administrator (DBA) can allocate and deallocate selected amounts of disk storage space to/from the IDBMS for the management of archived data sets and their respective catalogs.

8. Provide capabilities whereby the DBA can specify the disk packs on which specific catalogs or portions of catalogs will reside. It must be possible to remove the disk packs containing this information and replace them with other packs without rendering the system inoperable. When data has been removed in this manner, any request to access it must produce an information message to the user without rendering the system inoperable.

9. Provide capabilities which minimize required computer operator intervention and effort for the running of the IDBMS once it has been initiated.

10. Provide capabilities to save and restore selected data bases, to generate systemwide checkpoints of all data needed to restore the system to its operating status at the time the checkpoint was taken, and to roll back from checkpoints with restored data bases. (See 3.5(3)).

3.2 Performance

3.2.1 Data Accuracy

1. The IDBMS must be capable of storing, retrieving, and displaying data without loss of data precision or accuracy.

3.2.2 Validation

1. The system shall provide full error-checking and error-handling capabilities for user input errors and input data errors. This shall include: a) interactive, batch, and application program user syntax errors; b) validation of length and type (character, binary, etc.) of all information entered into the catalogs and other system internal data structures; c) validation of length and type of all data fields specified by interactive, batch, and application program users; and d) the ability to incorporate user-specified integrity checks (e.g., allowable values or range limits) to be applied against data fields input by a user or entered into catalogs or other system internal data structures.

3.2.3 Response Times

1. The system will have the following general usage characteristics:

- Data base/catalog search and retrieval operations will typically be performed more frequently then update operations.

- Most searches will involve multiple (2 to 10) keys.

- Most searches will result in the retrieval of multiple (typically 10 to 1,000) entries/records.

- Update operations will typically be performed by a more restricted group of users than retrieval operations.

- Update speed will typically be bound by the time required to construct the update information, e.g., to extract descriptors from a data set or from information entered at user terminals.

Therefore the system must be optimized for search and retrieval speed (vs. update speed) and for multi-key searches (vs. single-key searches).

2. Response time requirements are characterized by the following single-user scenarios which the system must accommodate:

a. Define a new data element to the system, including its name, length, and type of representation, within one second. Define a new data base to the system, including its constituent element names and their logical relationship, and initialize the data base within 3 seconds.

b. Initialize a data base, load records from a source file into the data base and build required indices at the rate of 100 records per second (2 minutes total) given the following characteristics:

Source file: sequential file of fixed length records. Record size: 80 bytes.
Number of records in source file: 10,000
Number of indices to be built: 10

c. Assuming that a data base intially has the following characteristics:

Number of records in data base: 2 million
Average record size: 80 bytes
Number of indices defined: 10

load records from the source file described under (b) into the data base and update required indices at the rate of 20 records per second (10 minutes total).

d. Perform an index search on the data base described under (c) and display a count of the number of records/entries meeting the search criteria within 2 seconds.

e. Retrieve selected fields from the data base described under (c) and output them to a sequential file at the rate of 100 records/entries per second. Retrieve and display these entries via a user terminal at the rate of 15 records per second.

3. When installed in the MSFC DBMS configuration, the IDBMS must be capable of constructing catalogs suitable for managing the storage of packetized data transferred to the DBMS at a rate of 100Mbps (see Section 3.1.2(2) and Reference 16).

3.2.4 <u>Flexibility</u>

A limited number of data sets and specific examples of the functions listed in Section 3.1 will be selected by GSFC personnel for initial system implementation. The initial system shall include all software needed to provide these selected functions for the selected initial data base. In addition the initial system shall be designed and implemented in a manner which allows for the easy expansion and incorporation of modules to support additional functions and additional data types. It is envisioned that this system growth will occur in a phased incremental manner. This potential system growth must be fully accounted for in the initial system design. The system must be flexible and easily expandable to accommodate:

1. Dynamic local archive and a dynamic collection of data cataloged by the system but not locally archived. This includes new data types and data formats as well as additions/deletions/replacements of data sets for existing data types and formats.

2. Changes (adds/modifications/deletes) to the type of information items stored in the catalog, changes to the level of data aggregation described as a single entity in the catalog, and changes to the information contained within the catalog.

3. Changes to data selection criteria and catalog search criteria.

4. Incorporation of new functions and data access and manipulation capabilities (e.g., subset extraction, averaging/regridding, sort/rearranging, plot/list/display, quality checks, new output tape formats, etc.) for existing data types.

5. Extension of existing system functions to handle new data types.

3.3 Inputs-Outputs

Data will be entered into the local archive via magnetic tape, disk, telecommunications, and IDBMS users (interactive, batch, and application program users) in a variety of formats. Typical examples of climate data are given in Figure 3.1. Formats of packetized data to be handled in the NEEDS DBMS are given in References 16 and 17.

Data will reside in the local archive on tape, disk, and AMM, or other direct access storage devices. Data will be output to local and remote alphanumeric CRT and graphics terminals, printers, plotters, image display terminals, computer compatible tape, disk, application programs, and remote processors (via a local network as described in Section 4.1) in various formats depending on the user's requirements. Some examples of typical output products are:

- Selected catalog information describing those portions of the data base which meet user-specified search criteria.

- Data plots and gridded arrays showing Earth location of available data and numbers of data values available in specified latitude/longitude grid regions.

- Gridded arrays of data values at various space and time grid intervals.

- Tabular listings of selected data qualified by time, parameter, etc.

- Contour plots of 2, 3 and 4-dimensional data sets.

- Data value plots (for example, average monthly value vs. month for each 10° latitude zone).

- Displays of selected data in raster format on image analysis terminals.

Selection parameters for output products will be input in one of the following ways:

- Interactively at local and remote alphanumeric CRT terminals through user-specified commands and/or responses to system-generated prompts.

- Procedurally by application programs through the argument list of IDBMS-supplied FORTRAN-callable routines.

Printed outputs of user retrieved data will be 50-100 pages/user/day with 4400-6600 characters/page. Two to five computer compatible tapes (1600 and 6250 bpi) of data will be produced daily for each user.

The IDBMS must be able to read and write records up to 32k bytes in length.

3.4 Data Characteristics

The IDBMS must manage a large number of multi-source data types including imaging and non-imaging satellite data and derived geophysical parameters. These data characteristics are found in climate-related data. The IDBMS will have to handle at least 10 climate parameters with 3 to 10 sources per parameter and ~ 3 types of data products per source, the result being 100 to 300 different types of data products. Typical examples of such climate data are given in Figure 3.1. Catalog and data archive volume

Figure 3-1

TYPICAL PARAMETERS AND DATA SOURCES*

PARAMETER	SATELLITE	INSTRUMENT
1. Sea Surface Temperature	NOAA-2,3,4,5	Scanning Radiometer (SR)
	TIROS-N	Advanced Very High Resolution Radiometer (AVHRR)
	NIMBUS-7	Scanning Multichannel Microwave Radiometer (SMMR)
2. Sea Ice Concentration	NIMBUS-5,6	Electrically Scanning Microwave Radiometer (ESMR)
	NIMBUS-7	SMMR
3. Ozone	NIMBUS-4	Backscatter Ultraviolet Experiment (BUV)
	NIMBUS-7	Solar Backscater Ultraviolet (SBUV) and Total Ozone Mapping Spectrometer (TOMS)
4. Clouds	NIBUS-5,6,7	Temperature Humidity Infrared Radiometer (THIR)
	SMS/GOES	Visible and Infrared Spin Scan Radiometer (VISSR)
5. Weather Variables	-----	National Meteorological Center Archives (NMC)
6. Radiation Budget	NIMBUS-6	Earth Radiation Budget (ERB) Experiment
	NIMBUS-7	ERB
7. Ocean Rainfall	NIMBUS-5,6	ESMR
	NIMBUS-7	SMMR
8. Ocean Surface Winds	NIMBUS-7	SMMR
	SEASAT	Scatterometer (SCAT), SMMR
9. Snow Cover	NOAA-2,3,4,5	SR
	TIROS-N	AVHRR
	NIMBUS-7	SMMR
10. Stratospheric Aerosols	NIMBUS-7	Stratospheric Aerosol Measurement (SAM II) Experiment
	AEM	Stratospheric Aerosol and Gas Experiment (SAGE)

*Archive can include all products (sensor radiances and in situ or averaged derived parameters) for each instrument.

Figure 3-2

ESTIMATED VOLUME OF CLIMATE-RELATED DATA

<u>ARCHIVE VOLUME ESTIMATE</u> (stated in equivalent 1600 BPI tape units):

100-300 data sets
1-300 tapes per data set (average)
10-100 files per tape (but sometimes over 1,000)

TOTAL ARCHIVE:
Result: 100-90,000 tapes, 1,000-9,000,000 tape files, 3×10^{10} - 2.7×10^{13} bits (at 3×10^{8} bits per tape)

<u>CATALOG VOLUME ESTIMATE</u> (stated in equivalent 1600 BPI tape units):

High Level Data Sets - Summary Description:

80 bytes per data set * 100-300 data sets = 8K-24K bytes

High Level Data Sets - Individual Detailed Description:

3K-8K bytes per data set * 100-300 data sets = 300K - 2.4M bytes

*Tape Level Catalog Information:

80 bytes per tape entry * 1-300 tapes per data set * 100-300 data sets = 8K - 7.2M bytes

*File Level Catalog Information:

40 bytes per file entry * 10-100 files per tape * 1-300 tapes per data set * 100-300 data sets = 40K - 360M bytes

TOTAL CATALOG: 350K - 370M bytes

*While the system must provide flexibility in specifying the level of data aggregation described as a single entity in the catalog, tape and file level aggregates are used here as typical examples for volume estimation purposes.

estimates for the climate-related data sets to be handled by the IDBMS are given in Figure 3.2.

Characteristics of packetized data, including the packet length fields which may be used to compute estimates of the volume of packet data to be managed by the MSFC DBMS are provided in References 16 and 17.

Examples of other Applications data which are currently utilized by existing discipline oriented systems and which may be cataloged and managed by the IDBMS are given in Appendix C.

3.5 Failure Contingencies

1. The system must provide full error-checking and error-handling capabilities. Errors include system I/O errors, system crashes, user aborts, and corruption or loss of system files.

2. The system must provide capabilities to periodically create tape backup copies of all (or selected) disk-resident system files (including catalogs, data and associated indexes) and to restore the system to this previous checkpoint by reloading from tape in the event of severe system corruption or degradation.

3. The system must maintain a system log file in which all changes to on-line files and catalogs and all save and restore operations are recorded so that the system can be rolled forward from a restored data base to its status as of some later point in the processing cycle (e.g., to just before a system crash). The system log must be entirely under the control of the system so that no user can read from or write to it directly.

4. The system must provide the capability to recover from all software/hardware problems occurring during system execution which leave the system in a corrupted state. Examples and the way they are to be handled are given below:

- System crashes, user aborts, or I/O errors which occur in the middle of an update to the catalog or other system files and which leave tables, pointers, indexes, etc. in an inconsistent state. The system will provide the capability to back out the partial update.

- I/O errors which occur while reading data tapes (to make catalog insertion, extract data subset, etc.). Depending on the nature of the error and type of function involved, the system will provide the capability to either recover and continue or to back out and re-do the partially completed transaction.

BIBLIOGRAPHIC DATA SHEET

1. Report No. TM83942	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Data Base Management System Analysis and Performance Testing with Respect to NASA Requirements		5. Report Date August, 1981
		6. Performing Organization Code 931
7. Author(s) E. A. Martin, R. V. Sylto, T. L. Gough, H. A. Huston, J. J. Morone		8. Performing Organization Report No.
9. Performing Organization Name and Address Information Management Branch NASA/Goddard Space Flight Center Greenbelt, MD 20771		10. Work Unit No.
		11. Contract or Grant No.
		13. Type of Report and Period Covered Technical Memorandum
12. Sponsoring Agency Name and Address NASA/Goddard Space Flight Center Greenbelt, MD 20771		
		14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

This study was conducted to evaluate several candidate Data Base Management Systems (DBMS) that could support the NASA End-to-End Data System's Integrated Data Base Management System (IDBMS) Project which was later rescoped and renamed the Packet Management System (PMS). The candidate DBMS systems which had to run on a VAX 11/780 computer system were ORACLE and RIM, both based on the relational data base model, and SEED, a CODASYL network approach. Goals of the study included assess capability to manage large amounts of data (i.e. at least a million input records), determine ingestion rates, measure the efficiency of the various data access techniques and to evaluate qualitative characteristics. The load results indicate that SEED is significantly faster than RIM and ORACLE below a half million records. From a half million to a million records the incremental load rates begin to favor ORACLE and at over one million records, ORACLE is clearly superior. Each system varied significantly in the amount of space required. While the results indicate that none of the candidate systems fully meet the original requirements of the IDBMS, they do indicate that these systems could be used as a central core to an IDBMS-like system around which additional software would have to be built to satisfy many of the requirements.

17. Key Words (Selected by Author(s)) Data Base Management Systems DBMS Performance Testing ORACLE SEED, RIM		18. Distribution Statement	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages	22. Price*

*For sale by the National Technical Information Service, Springfield, Virginia 22161

GSFC 25-44 (10/77)